UNIVERSIDADE LISBOA
FACULDADE CIÊNCIAS
DEPARTAMENTO DE FÍSICA

Numerical Simulations in Micromagnetism

Pedro Jorge Andrade Matos

Mestrado em Física
Física da Matéria Condensada e Nanomateriais

Dissertação Orientada por:
Thomas Gasche
José Pires Marques

2017

Agradecimentos

Esta dissertação vem em resultado de uma batalha sistemática para compreender a teoria, e de imaginar, construir e testar os algoritmos aqui apresentados, marcando o periodo de maior apredizagem que tive. Tal não teria sido possível sem as muitas oportunidades que me foram dadas pelas pessoas que me rodeiam.

Gostaria em primeiro lugar de agradecer ao professor Thomas Gasche, por me ter dado o tempo necessário, a liberdade de pensar e a possibilidade de cometer erros.

Agradeço ao professor Luís Bento por me ter ajudado a resolver alguns detalhes teóricos no Capítulo 4. E também ao professor José Pires Marques e às professoras Patrícia Faísca, Margarida Godinho, Margarida Cruz e Iveta Pimentel por terem estado disponíveis quando foi importante.

Finalmente, agradeço ao meu pai e a minha mãe que sempre me apoiaram em tudo.

Abstract

In this thesis we start by reviewing theoretical aspects of micromagnetism. Since many technological applications only depend on the behavior of magnetization of ferromagnets with the applied magnetic field at a macroscopic scale, there is no need to use a highly detailed theory like quantum mechanics. Micromagnetics is the framework that captures at a mesoscopic level the essential dynamical behavior of a magnetization field. It describes the combination of a very fast processional motion and a slower damping toward the magnetic field.

The two central relations are the Landau-Lifshitz (LL) and Landau-Lifshit-Gilbert (LLG) equation's. We show how to derive the first assuming two main experimental and theoretical observations: (1) the local magnetization norm is conserved; (2) the equilibrium state that both the magnetic field and magnetization aligned. We then analyze the dynamics implied by four magnetic field contributions: the applied field; the anisotropy field which has a similar behavior to an applied field along the lattice axis; the stray field which depends on all other magnetic moments; the exchange field which is the most relevant term, it tends to smooth out the magnetization direction.

We then introduce the LLG equation by representing the damping term by Rayleigh Dissipation. It is an implicit equation of the magnetization. Our goal is to develop a Python code to integrate this equation. We start then by combining it with the Finite Element Method to discretize space and the Implicit Midpoint Rule to discretize time. To avoid meshing the surroundings of the system that was required for introducing the asymptotic boundary conditions for the calculation of stray field potential, we use the Boundary Element Method and a new potential to restrict these calculations to the system. Using the Newton Raphson Method we obtain a linear system of equations that is solved at each time step yielding the evolution of the magnetization.

Setting our code to solve a standard problem for permalloy block $120 \times 120 \times 10\,\mathrm{nm}^3$ we compare our results of magnetization evolution with those of the OOMMF micromagnetic simulator to validate our code. The results of our code compare very well with those of the OOMMF simulation, specifically the time evolution of y component of magnetization, its Discrete Fourier Transform as well as the spatial distribution of the amplitudes of the Fourier coefficients for two distinct resonance frequencies.

Keywords: Micromagnetism; Landau-Lifshitz; Landau-Lifshitz-Gilbert; Rayleigh Dissipation; Implicit Midpoint Rule; Finite Elements Method; Newton-Raphson Method; Boundary Element Method; Discrete Fourier Transform; Python

Resumo

Ao nível quântico, os materiais ferromagnéticos são caracterizados por magnetismo resultante do spin dos eletrões cuja descrição completa requer a aplicação da Mecânica Quântica. Estes materiais quando $T \ll T_c$ são caracterizado por domínios *i.e.* regiões de momentos magnéticos com a mesma direcção e sentido, que são o resultado de fortes interações de troca. A orientação da magnetização local varia à escala destes domínios quando o sistema está em equilíbrio. Quando um campo magnético externo é aplicado sobre o sistema, a magnetização de cada domínio tende a alinhar com este campo. Se o campo for suficientemente forte passamos de múltiplos domínios, a um único monodomínio. Quando este campo é desligado, estes voltam a formar-se mas com orientações diferentes das iniciais. Este comportamento da magnetização total do ferromagnete em função do campo aplicado designa-se por curva de histerese e a sua taxa de variação quando o campo é zero determina a estabilidade da magnetização total. É o formato desta curva que permite aplicações tecnológicas de gravação e leitura de dados em discos rígidos, construção de eletroímanes ou libertação de calor para destruir células cancerosas.

Com o objetivo de aplicar estes materiais é preciso traçar a curva de histerese. Observa-se que é uma relação entre variáveis na escala macroscópica. Conclui-se assim que uma descrição quântica do material seria desnecessariamente rigorosa. O que é necessário é informação a uma escala mesoscópica que capture as características observadas quanticamente e que reproduza o comportamento magnético a uma escala macroscópica. A área de trabalho que estabelece esta ligação é a de Micromagnetismo. Ao invés de partir dos momentos magnéticos associados a electrões, divide-se a amostra em subvolumes que são grandes quando comparados com a constante de rede do ferromagnete, mas muito menores que o tamanho da amostra. A informação de como a magnetização varia no interior destes subvolumes é desprezada. Em relação à dinâmica da magnetização duas equações foram desenvolvidas uma por Landau e Lifshitz, e outra por Landau, Lifshitz e Gilbert. Estas equações descrevem a combinação de dois movimentos do vector magnetização e dependem do campo magnético local. Um movimento rápido de rotação da magnetização em torno da direcção do campo magnético e outro movimento mais lento de amortecimento que descreve a perda de energia do sistema. A evolução temporal deixa de ocorrer quando a magnetização está alinhada com o campo magnético local, obtendo-se finalmente uma configuração mesoscópica que está associada a um ponto da curva de histerese.

Nesta dissertação iremos rever primeiro como obter a equação de Landau-Lifshits (LL) a partir de duas suposições: (1) a conservação da norma do vector de magnetização local; (2) o estado de equilíbrio é atingido quando magnetização e campo estão alinhados. A seguir

apelando à mecânica clássica, Gilbert propôs representar o termo de amortecimento via dissipação de Rayleigh obtendo-se assim a equação de Landau-Lifshits-Gilbert (LLG).

Nestas duas equações o campo magnético local apresenta quatro contributos: campo aplicado; anisotropia; campo desmagnetizante; campo de troca. No equilíbrio final observa-se o alinhamento entre magnetização e campo magnético.

Ambas as duas hipóteses acima são válidas para todos os contributos magnéticos, a diferença fundamental só está na fonte do campo. O campo aplicado é determinado externamente. O termo de anisotropia é representado por um campo ao longo da estrutura cristalina do material. O campo desmagnetizante resulta da soma dos campos de todos os outros momentos magnéticos. O termo de troca ferromagnético tende a orientar momentos magnéticos vizinhos no mesmo sentido.

Começamos com a equação de LLG, uma equação implícita na magnetização, e o campo desmagnetizante determinado pela equação de Poisson. Sobre o sistema de equações LLG e equação de Poisson aplicamos o "Implicit Mid-Point Rule", discretizando as derivadas temporais sob um certo incremento de tempo e substituindo a magnetização e o potencial por médias entre estes dois instantes de tempo. Obtém-se assim duas equações que os relacionam e que correspondem a um sistema implícito sobre o próximo estado em função da magnetização e do potencial no estado anterior ou inicial.

Dada a complexidade destas equações e o facto de o incremento temporal ser pequeno, a magnetização e o potencial não variam muito neste intervalo, donde deixa de ser necessário informação sobre variações de ordem superior à primeira. Expandindo os residuos destas duas equações até primeira variação do campo e do potencial, obtém-se como coeficientes os Jacobianos dos residuais. Este é o método de Newton-Raphson e é muito usado para resolver equações com escalas de tempo muito diferentes, como observado nas equações LL e LLG. Com este método define-se o residuo associado à equação LLG e à equação de Poisson e procura-se a solução do sistema de equações correspondente.

Para implementar computacionalmente a resolução destas equações que são contínuas no espaço é preciso discretiza-las utilizando-se o Método de Elementos Finitos. O sistema e a sua vizinhança são divididos numa malha de tetraedros e a cada um dos vértices de cada tetraedro associa-se um vector com as três componentes de magnetização e o potencial magnético. Supondo que são funções lineares é possível representá-las como combinação linear de funções de base também lineares e que dependem da localização dos vértices de cada tetraedro. Desta forma o sistema de equações dos resíduos pode ser simplificado, mas devido à presença de segundas derivadas espaciais nos termos de troca e de "Poisson" é essencial reduzir a ordem destas equações. Para tal usa-se a forma fraca do sistema de residuos expandidos em primeira ordem. O problema inicialmente contínuo passa a um sistema de equações lineares em que a incógnita é um vector cujas componentes são todos os incrementos de magnetização e potencial em todos os vértices da malha e com o qual é possível actualizar a configuração em cada instante.

Para evitar ter de usar uma malha para o reservatório (espaço no exterior à amostra) iremos introduzir um novo potencial que satisfaz uma nova equação de Poisson que tem condições de fronteira de Neumann na fronteira do sistema. Com o Método Elementos de Fronteira é possível mapear a solução deste problema na fronteira para o potencial do campo desmagne-

tizante também na fronteira. Desta forma conseguimos substituir as condições de fronteira assimptóticas que originalmente requeriam a presença do reservatório, por um problema de Poisson com condições de fronteira de Dirichlet. O custo desta reformulação do cálculo do potencial do campo desmagnetizante é o aumento do sistema de equações a resolver e o cálculo da matriz que mapeia estes potenciais, que é densa e computacionalmente exigente de ser obtida. Contudo as suas entradas são constantes donde este bloco do sistema de equações só precisa de ser calculado uma vez.

Cada um dos outros termos do campo magnético também irá contribuir com o seu Jacobiano. Após introdução de uma quadratura nodal para aproximar os integrais aí presentes obtém-se a forma final para ser implementada computacionalmente.

Com estas expressões e os dois problemas de Poisson é possível calcular as entradas da matriz. Dado que tal tem de ser feito a cada incremento temporal o processo de cálculo e construção da matriz tem de ser rápido. Daí que tenhamos desenvolvido os algoritmos em Python para cada termo de LLG usando operações vectoriais. O sistema de equações pode agora ser resolvido o que permite actualizar um estado corrente de magnetização e dois potenciais ao longo de todo o sistema magnético para um estado seguinte. Constrói-se a história do sistema magnético, donde cálculos da sua curva de histerese são possíveis e assim, finalmente desenvolver as suas aplicações.

Na parte final deste dissertação para validar o nosso código implementámos a resolução de um problema standard sobre um bloco de permalloy de $120 \times 120 \times 10\,\mathrm{nm}^3$. Obtivemos resultados para a evolução da componente y da magnetização e da Transformada de Fourier Discreta, assim como a distribuição espacial das amplitudes e fases associados a duas frequências de ressonância distintas. Estes cálculos são comparados com os resultados publicados para o mesmo problema, concluindo-se que estão em boa concordância.

Palavras-Chave: Micromagnetismo; Landau-Lifshitz; Landau-Lifshitz-Gilbert; Dissipação de Rayleigh; Método Elementos Finitos; Método Newton-Raphson; Método Elementos de Fronteira; Transformada de Fourier Discreta; Python

Contents

List of Figures

Abbreviations

CG - Coarse Grain
FD - Finite Differences
FE - Finite Elements
IMR - Implicit Mid-Point Rule
LL - Landau-Lishitz
LLG - Landau-Lifshitz-Gilbert
NR - Newton-Raphson
BEM - Boundary Element Method
DFT - Discrete Fourier Transform
RHS - Right Hand Side
LHS - Left Hand Side
BC - Boundary Conditions
MC - Matching Conditions

Chapter 1

Introduction

At the fundamental structure of materials we observe the orbital angular momentum of electrons and spin angular momentum of electrons [1]. Orbital magnetism is associated with *diamagnets* and *superconductors*, while electron spin is associated with paramagnets and ferromagnets [2]. In classical electromagnetism all of these are uniquely represented by a field $\mathbf{J}$, that is interpreted at every spatial point $\mathbf{x}$ as indicating how much charge crosses a given area element per unit time [3]. The way these charged currents affect themselves is represented by a magnetic field, $\mathbf{H}$. The framework that describes this interaction is in such a way that other representations of these currents are more adequate. Instead of charged currents, it is possible to define the magnetization field, $\mathbf{M}$, that also takes into account the geometry of the currents.

The way charged particles affect each other is reformulated as describing how the magnetization field in one region affects other regions. The dynamics that comes as a result of this interactions we call the process of *magnetization* of materials.

All materials exhibit magnetization at some scale. In this thesis we will focus on a subset of these materials known as ferromagnets. These respond to externally applied magnetic fields in more complicated manner than the simple relation $\mathbf{M} = \chi \mathbf{H}_{ap}$, and have a relation not represented by a closed function but by a non-singular and multivalued relation represented by a plot and designated curve of *hysteresis*, [3, p. 421, 443].

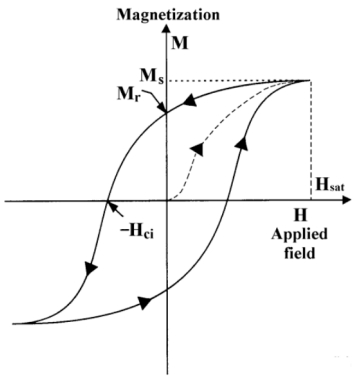

Figure 1.1: Hysteresis curve for a ferromagnetic material represents how the total magnetization $\mathbf{M}$ of the sample behaves as we change the applied field, [3].

It is a macroscopic property that can be experimentally obtained with great precision using a *Superconducting Quantum Interference Device* [4]. The macroscopic magnetization of the entire sample is given as a function of applied field. The causes of such behaviour can be understood using a mesoscopic approach. Introducing the existence of magnetic domains in the sample, that are subparts of the system where magnetization direction and norm are independent of $\mathbf{x}$, the variation occurs only between domains. The existence of magnetic domains can be seen in the following way: Exchange interaction drive the entire sample into a single domain, but if we introduce interactions like stray field, this single domain is broken in smaller ones. The corresponding field can be treated analogously to an electric field, defining fictitious magnetic charges, with volume density given by $\rho^* = -\nabla \cdot \mathbf{M}(\mathbf{x})$ and a surface density given by $\sigma^* = \mathbf{M}(\mathbf{x}) \cdot \hat{n}$ where $\mathbf{M}(\mathbf{x})$ is the magnetization at $\mathbf{x}$. In this way the monodomain in figure 1.2 second picture would have $\rho^* = 0$ in its interior but the top surface would have "positive" charge while the bottom have the opposite.

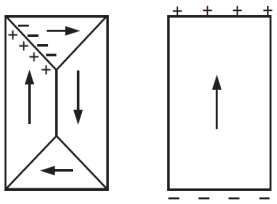

Figure 1.2: Two domain configurations. The first one has a lower energy [3].

Apelling to arguments from statistical physics, the configuration in figure 1.2 first picture would have a lower energy since the charges cancel at domain boundaries and $\mathbf{M}$ is orthogonal to the surface of the system, hence energy associated with stray field is zero and this configuration is the one observed. However in real materials, it is observed that the existence of these domains also depends of the presence of strong anisotropy effects in the material [5], *i.e.*, preferable directions of alignment within the material.

In figure 1.1 we see at the beginning of the path, when all domains are disaligned, $|\mathbf{M}| = 0$, corresponding to first picture of figure 1.2. This configuration minimizes the sum of exchange energy and stray field energy.

However if we introduce an applied field, the magnetization will tend to align with this field $\mathbf{H}_{ap}$. The domains in line with this field grow progressively at the expense of the disaligned ones as they rotate toward the magnetic field. After a given intensity, the sample is only one domain, second picture in figure 1.2.

Now, if the field is reduced to zero, the magnetization will not retrace through the previous path, instead a new state is reached where $|\mathbf{M}_r| \neq 0$, a remanescent magnetization prevails like in permanent magnets. Domain emerge again but not the same as in the beginning of this path. The new magnetization configuration retain some of the properties that characterize recent states, in this case the state where all magnetization was aligned with the applied field. Continuing applying the magnetic field in opposite direction, the magnetization will reach zero at $-\mathbf{H}_{ci}$ and finally if we continue increasing the magnetic field in the opposite sense, the entire sample will align in the opposite direction. Inverting the process we can close the cycle.

Notice during the entire process the magnetization norm, $|\mathbf{M}(\mathbf{x})|$, inside each domain is constant.

The behavior of domains shown by the hysteresis loop, allows very important applications.

- Consider then the following simple idea. With a small set of ferromagnetic samples. Each one will have their own hysteresis curves and under an applied external field each one will behave according to it. If we magnetize one sample in the y direction, after turning off the field, the monodomain disappears and several domain appear but in average they will point in the y direction. This configuration saved information in the most elementary form, we can call it 1. If we assume that magnetizing in the opposite direction means 0, then we can easily see that with a set of these materials we can encode information in binary format. The information can later be retrieved. Since magnetic moments point on average in one direction previously decided, they generate a magnetic field in that direction which can be measured. For example with materials whose resistance varies with the direction of the field, the field direction can be converted into one of current intensity within the measuring device.

  This is the main idea behind hard drive recording and reading, the direction of the magnetic moments on the hard drive encode the information we introduce. The guarantee that this information is saved through time can also be seen in the hysteresis curve. If we have an high H_{ci}, the curve will be very wide, so the derivative of magnetization with the applied field at $H = 0$ is close to zero, for the other small fields present within the hard drive and even external ones the magnetization will not change much, and the information is preserved.

- Ferromagnetic materials are also used in boosting the intensity of magnetic fields. Suppose a coil of electrical wire. Under a given current, a magnetic field is generated, if at the center of this coil this material is introduced, then it will be magnetized, the magnetic field produced by this sample will add to the field produced by the coils. The electromagnet is turned on to attract and must reach $-H_{ci}$ to the electromagnet be off.

  Since the energy spent is given by

$$\delta w = \mu_0 \int_V \mathbf{H}_{ext} \cdot \delta\mathbf{M}\, d^3x \tag{1.1}$$

  we can see that if H_{ci} is close to zero [3], the energy cost is lowered. A thinner hysteresis curve is cheaper!

- If to the work necessary to magnetize a sample to $\delta\mathbf{M}$, we subtract the variation of exchange energy, we obtain the heat produced by the material [6, p. 180]

$$\delta Q = \mu_0(H_{ext} + n_w M)\delta M \tag{1.2}$$

  Aligning the domains imply heat production. This can be used with medical purposes to eliminate tumors. Developing particles with receptors for these cancerous cells, after

they attach we apply an external magnetic field. The heating produced by the particles is transferred to the surrounding cells destroying them [7].

These examples show us not only that initial and final states of equilibrium magnetization are important, like we also saw in the hysteresis loop, but the dynamics of relaxation is important as well if we want to understand how to design materials with new hysteresis shapes.

Still within a mesoscopic framework we can build a theory that explain the dynamics of magnetization. Proposed by Landau and Lifshitz in his paper [8] and later reformulated by Gilbert in [9] these phenomenological equations constitute the foundations of Micromagnetism.

In this thesis we first introduce the theoretical background for Landau-Lifshitz equation in Chapter 2 and for the remaining Chapters we intent to develop the code to integrate the Landau-Lifshitz-Gilbert equation by applying the Finite Element Method and Newton-Raphson Method giving us the magnetization dynamics.

Chapter 2

Landau-Lifshitz and Landau-Lifshitz-Gilbert Equations

Micromagnetism is the theory that has the goal of using classical electromagnetism to model the behaviour of magnetic materials. As the name suggests, it describes the material at the micrometer scale.

On the atomic scale, magnetization is a result of charged currents associated to spin and atomic orbital moments whose quantum description is highly complicated. However approximations can be made if the final output of the theory does not require all the microscopic details. For ferromagnetic materials it is known that at $T \ll T_c$, the system's magnetic structure is characterized by domains of aligned magnetic moments due to strong exchange quantum interactions that with crystaline anisotropy have the tendency to be oriented along some lattice directions of the sample [2].

Under these conditions a detailed quantum description is unnecessary, it is then open a strategy of coarse graining that has the goal of simplifying the description of the magnetic moments and the dynamic laws they obey [11].

In what follows the dynamics of an out of equilibrium system of magnetic moments subjected to a given magnetic field will be derived from two fundamental principles giving us LL equation, then from energy considerations, the magnetic field contributions are obtained through variational calculus, the final equilibrium point condition is derived. With the LL equation each of the magnetic field contribution has dynamical implications that will be analysed. Finally a second formulation of LL equation is introduced, Landau-Lifshitz-Gilbert equation, which describes a dynamics similar to the first and that we will use in subsequent Chapters.

2.1 Landau-Lifshitz equation

Each volume element is characterized by a current $\mathbf{j}_M$ which is described by quantum mechanics and is due to the motion of charges [3, p. 408-410]. To simplify we define $\mathbf{M}(\mathbf{x})\Omega$ to be the average of magnetic moments in the vicinity of $\mathbf{x}$ in a cubic cell of volume, Ω, of side

much bigger than the lattice spacing but much smaller than the sample size, it is given by

$$\mathbf{M}(\mathbf{x}) = \frac{1}{\Omega}\mathbf{m} = \frac{1}{2\Omega}\int_{\Omega} \mathbf{s} \times \mathbf{j}_M \, d^3 s \tag{2.1}$$

where $\mathbf{m}$ is the total magnetic moment associated to volume Ω.

Once the magnetization is known, the details of how it is generated are irrelevant. Despite having an explicit relation, its computation will not be needed. It is a continuous function that will not have variations on scales of the order of lattice spacing, the details of how variations occur within the sample subdivision don't belong to micromagnetism, instead the theory will focus in determining how variations in $\mathbf{M}$ on the scale larger than this sample subdivision evolve in time when the system is out of equilibrium.

Following the coarse graining strategy (CG) used above for magnetization, its fundamental dynamical processes can also be recast in this new scale. Despite the fact that $\mathbf{j}_M$ comes from spin and orbital magnetic moments [3, p. 409], when we use $\mathbf{m}$ or $\mathbf{M}$ defined in 2.1 both this sources are irrelevant, therefore to understand the dynamics of $\mathbf{m}$ we can use a simple example from classical electromagnetism. Suppose a current distribution smaller than a lattice cell and that satisfies $\nabla \cdot \mathbf{j} = 0$, for example a circular loop of current. The following identity then follows

$$\nabla \cdot (x_k x_l \mathbf{j}) = x_l j_k + x_k j_l \tag{2.2}$$

where k and l indices indicate the components of the position vector $\mathbf{x} = (x, y, z)^T$.

Using the divergence theorem we find

$$\int d^3x \, j_k x_l = -\int d^3x j_k x_k \tag{2.3}$$

from which we can derive

$$\int d^3x \, j_k x_l = \frac{1}{2}\int d^3x \, (j_k x_l - j_l x_k) \tag{2.4}$$

If the current $\mathbf{j}$ is subjected to a magnetic field $\mathbf{B} = \mu_0(\mathbf{M} + \mathbf{H})$ then the torque is given by

$$\mathbf{N} = \int d^3x \, \mathbf{x} \times (\mathbf{j} \times \mathbf{B}) \tag{2.5}$$

We would like to reformulate it in terms of magnetic moment defined in (2.1). Since $\mathbf{x}$ and $\mathbf{j}$ are already present we only need to join them. Using the identity

$$\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) + \mathbf{b} \times (\mathbf{c} \times \mathbf{a}) + \mathbf{c} \times (\mathbf{a} \times \mathbf{b}) = 0 \tag{2.6}$$

we get

$$\mathbf{N} = -\int d^3x \, \mathbf{j} \times (\mathbf{B} \times \mathbf{x}) - \int d^3x \, \mathbf{B} \times (\mathbf{x} \times \mathbf{j}) \tag{2.7}$$

Introducing now $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = \mathbf{b}(\mathbf{a} \cdot \mathbf{c}) - \mathbf{c}(\mathbf{a} \cdot \mathbf{b})$ yields

$$\mathbf{N} = -\int d^3x \, \mathbf{B}(\mathbf{j} \cdot \mathbf{x}) + \int d^3x \, \mathbf{x}(\mathbf{j} \cdot \mathbf{B}) - \int d^3x \, \mathbf{x}(\mathbf{B} \cdot \mathbf{j}) + \int d^3x \, \mathbf{j}(\mathbf{B} \cdot \mathbf{x}) \tag{2.8}$$

The first integral will be zero for a circular loop of current as $\mathbf{j}$ will be perpendicular to $\mathbf{x}$. For arbitrary shape current distributions, since we assumed they are small, we can consider $\mathbf{B}$ constant and factor it. The first integral can then be shown to be also be zero using (2.2) and the divergence theorem

$$\int d^3x\, j_x x + j_y y + j_z z = \frac{1}{2}\int_S d\mathbf{S}\cdot\mathbf{j}(x^2+y^2+z^2) = 0 \tag{2.9}$$

where S encloses the current and therefore $\mathbf{j}=0$. The torque is reduced to the last integral[1] which is reformulated by components in terms of the magnetic moment using (2.4) and where factoring $\mathbf{B}$ yields

$$N_k = B_l \int d^3x\, j_k x_l = B_l \frac{1}{2}\int d^3x\,(j_k x_l - j_l x_k) = \epsilon_{kli} B_l m_i \tag{2.10}$$

These components define now the torque as

$$\mathbf{N} = \mathbf{m}\times\mathbf{B} \tag{2.11}$$

We can relate the torque with magnetic moment by first relating it with the angular momentum. Suppose the current $\mathbf{j} = \sum_k q_k \mathbf{v}_k \delta(\mathbf{x}-\mathbf{x}_k)$, then substituting in (2.1) gives us

$$\mathbf{m} = \frac{1}{2}\sum_k q_k(\mathbf{x}_k\times\mathbf{v}_k) = \sum_k \frac{q_k}{2m_k}\mathbf{L}_k = \frac{q}{2m}\mathbf{L} \tag{2.12}$$

where the angular momentum of the particle k is given by $\mathbf{L}_k = \mathbf{x}_k\times(m\mathbf{v}_k)$ and the last step assumes all k particles are equal and the total angular momentum is $\mathbf{L}=\sum_k \mathbf{L}_k$. This simple relation allows the reformulation of (2.11) in terms of the magnetic moment of the current

$$\frac{d\mathbf{m}}{dt} = \gamma\mathbf{m}\times\mathbf{B} \tag{2.13}$$

where we define the gyromagnetic ratio[2] as $\gamma = \frac{q}{2m}$. If initially the current magnetic moment is not parallel with the magnetic field and $q>0$ then it will precess around it in a clockwise way and will never align with it. Observing now that $\frac{d}{dt}$ and $\times\mathbf{B}$ operations are linear, our CG strategy can again be applied. Since a current distribution is described by this equation, so each one of the many current in a volume Ω will have dynamics described by a similar equation, whose net effect is

$$\begin{aligned}
\frac{1}{\Omega}\sum_j \frac{d\mathbf{m}_j}{dt} &= \frac{1}{\Omega}\sum_j \gamma\mathbf{m}\times\mathbf{B} \\
\frac{d\frac{1}{\Omega}\sum_j \mathbf{m}_j}{dt} &= \gamma\left(\frac{1}{\Omega}\sum_j \mathbf{m}\right)\times\mathbf{B} \\
\frac{d\mathbf{M}}{dt} &= \gamma\mathbf{M}\times\mathbf{B}
\end{aligned} \tag{2.14}$$

[1] Observe that (2.4) can be used to show that $\int d^3x\,\mathbf{x}(\mathbf{B}\cdot\mathbf{j}) = -\mathbf{m}\times\mathbf{B}$.

[2] If q is the electron charge, $q=|e|$, then $\gamma\to\gamma_e$, the electron gyromagnetic ratio.

which can be rewritten in terms of $\mathbf{H}$ using $\mathbf{B} = \mu_0(\mathbf{M} + \mathbf{H})$ giving us

$$\frac{d\mathbf{M}}{dt} = \gamma\mu_0 \mathbf{M} \times \mathbf{H} \tag{2.15}$$

At the scale of each cell the magnetization $\mathbf{M}$ evolves in time by *precessing* around the local magnetic field as well. However it is a experimental fact that eventually the magnetization will align with the field, and also that for temperatures below T_c its norm is conserved [3, 11]. Rather than derive a *damping* term from fundamental quantum processes, with no more knowledge than these two principles we can look for the simplest dynamical expression that would be compatible with them [11], and then test its consequences against experiment. Suppose then a first order dynamical equation that depends on the magnetization and the field, $\mathbf{H}$:

$$\frac{d\mathbf{M}}{dt} = \mathbf{V}(\mathbf{M}, \mathbf{H}) \tag{2.16}$$

Given that the norm is constant, there is no variation along the magnetization vector, so it must be an important direction, hence we construct an orthogonal reference frame that includes it: $\{\mathbf{M}, \mathbf{M} \times \mathbf{H}, \mathbf{M} \times (\mathbf{M} \times \mathbf{H})\}$. By representing $\mathbf{V}$ in it, the equation becomes

$$\frac{d\mathbf{M}}{dt} = a_0\mathbf{M} + a_1\mathbf{M} \times \mathbf{H} + a_2\mathbf{M} \times (\mathbf{M} \times \mathbf{H}) \tag{2.17}$$

Since the norm of $\mathbf{M}$ does not change in time, then a_0 is zero. The second principle states that an equilibrium magnetization configuration points along the magnetic field, that is $\mathbf{M}_{eq} \times \mathbf{H} = 0$, but that is readily satisfied by the equation because then $\frac{d\mathbf{M}}{dt} = 0$.

What is left is the determination of a_1 and a_2, which must include μ_0 as we are expressing the dynamics equation in terms of $\mathbf{H}$ instead of $\mathbf{B}$. The sign of the first must negative if the charge is $q = -e$ in (2.15), the second must also be negative by noting that then this second term updates the magnetization towards the magnetic field. From (2.15) the constant a_1 will be the electron gyromagnetic ratio times the vacuum permeability, $\gamma_L = \gamma_e\mu_0$, and it controls the rate of precession of magnetization, it has units $\frac{m}{As}$. The constant a_2 is decomposed as $\frac{\alpha\gamma_L}{M_s}$ [11, p. 26], where α is a small dimensionless damping constant usually between $10^{-4} - 10^{-2}$.

The equation obtained is the Landau-Lifshitz equation

$$\frac{d\mathbf{M}}{dt} = -\gamma_L\mathbf{M} \times \mathbf{H} - \frac{\alpha\gamma_L}{M_s}\mathbf{M} \times (\mathbf{M} \times \mathbf{H}) \tag{2.18}$$

It describes the continuous dynamics of a magnetization function subjected to a magnetic field, characterized by precession about $\mathbf{H}$ and damping that represents the thermal contact with the reservoir. Notice however that nothing was assumed about the source of such field. Several contributions will be taken into account: an externally applied field $\mathbf{H}_{ap}$ generated by the reservoir; the influence of lattice structure directions, $\mathbf{H}_{AN}$; the demagnetizing field $\mathbf{H}_M$; and the microscopic representation of quantum exchange $\mathbf{H}_{EX}$. The total magnetic field in LL equation (2.18) is

$$\mathbf{H}_{tot} = \mathbf{H}_{ap} + \mathbf{H}_{AN} + \mathbf{H}_M + \mathbf{H}_{EX} \tag{2.19}$$

In the next section each contribution to the dynamics and energy will be analysed individually.

2.2 Energy and Dynamics of Magnetization

The energy associated with the system under a given field contribution will be given. For both the external field and stray field the energy is readily obtained from the fields, for anisotropy and exchange, the energy function form is first introduced and from it, through variational calculus, its field representation is obtained. The implication of each magnetic field on the dynamics will be studied on next sections. But first it will be useful to simplify all the equations by identifying the natural scales for each quantity, it is possible then to put both the LL and energies in adimensional form.

Fundamental scales in Micromagnetism

The dynamical equation (2.18) contains parameters with units, with these and its combinations we can construct the fundamental scales for this particular problem. Re-expressing all quantities as adimensional will isolate its mathematical form, which has no units, and is therefore more simple.

The magnetization, $\mathbf{M}$, has norm M_s and units $\frac{A}{m}$, so it can be written as $\mathbf{M} = M_s\mathbf{m}$, where the new field[3] $\mathbf{m}$ point in the same direction as the original, but now its norm is 1. The magnetic field whose norm can change has the same units as magnetization and will be rescaled by M_s as well, $\mathbf{H} = M_s\mathbf{h}$. The spatial characteristic length has to be build by a combination of variables. Knowing that exchange interaction is characterized by a parameter A of units Jm^{-1} and that $\mu_0 M_s^2$ has units of Jm^{-3}, then defining $l_{EX} = \sqrt{\frac{2A}{\mu_0 M_s^2}}$ will have units of m, where the μ_0 is the vacuum permeability. Therefore we can measure lengths as $x = ul_{EX}$. For a generic function $f(x)$ spatial derivatives will be given by

$$\frac{df(x)}{du} = \frac{df(ul_{EX})}{du} = \frac{df(x)}{dx}\frac{dx}{du} \tag{2.20}$$

Which can be rewritten as

$$\frac{df(x)}{dx} = \frac{1}{l_{EX}}\frac{df(ul_{EX})}{du} \tag{2.21}$$

Gradients and Laplacians are composed by derivatives, then

$$\nabla_{\mathbf{u}} f(x,y,z) = l_{EX}\nabla_{\mathbf{x}} f(x,y,z) \tag{2.22}$$

$$\nabla^2_{\mathbf{u}} f(x,y,z) = l^2_{EX}\nabla^2_{\mathbf{x}} f(x,y,z) \tag{2.23}$$

From the units of $\mu_0 M_s^2$, introducing now the volume l_{EX}^3, energy scale is given by $E = e\,\mu_0 M_s^2 l_{EX}^3$, where e is dimensionless. Any energy term can be put in this form.

As we shall see in Chapter 3, the demagnetizing field has a potential formulation that satisfies a Poisson's equation $\nabla^2\phi = \nabla\cdot\mathbf{M}$ which must be adimensional as well. Defining the unit of potential, Φ, we have $\phi = \phi^{ad}\Phi$. Substituting in Poisson's equation together with (2.22) and (2.23) we have

$$\frac{\Phi}{l_{EX}^2}\nabla^2\phi^{ad} = \frac{M_s}{l_{EX}}\nabla\cdot\mathbf{m} \tag{2.24}$$

which suggests that the fundamental unit of potential is $\Phi = M_s/l_{EX}$.

[3]This $\mathbf{m}$ is not the magnetic moment we defined in (2.1). $\mathbf{m}$ is now defined as $\frac{\mathbf{M}}{M_s}$.

Finally the fundamental unit of time is obtained by combining the precession constant γ_L with the magnetization norm, M_s, giving us $(\gamma_L M_s)^{-1} = (\gamma_e \mu_0 M_s)^{-1}$ with units of s.

With these scales, substituting the (2.21) version for time unit into the LL equation (2.18), and factoring the fundamental units out of magnetization and magnetic fields the adimensional LL is

$$\frac{d\mathbf{m}}{dt} = -\mathbf{m} \times \mathbf{h} - \alpha \mathbf{m} \times (\mathbf{m} \times \mathbf{h}) \tag{2.25}$$

Not only it is cleaner, it brings attention to the important fact to the magnitude difference that separates the precession and damping terms. While the first has coefficient 1, the second is determined by α, which is 2 to 4 orders of magnitude smaller, the rotation motion is much faster than the damping. So huge is the difference between this two behaviour that differential equations that exhibit this kind of difference have their own name, stiff equations. Each contribution for $\mathbf{h}$ is now analysed where we assume $\mathbf{h}$, $\mathbf{m}$ and x are dimensionless.

Contribution 1: The field generated by the reservoir

If a magnetic material is subjected to an applied field $\mathbf{h}_{ap}$ then the energy will be given by

$$e_{ap} = -\int_V \mathbf{h}_{ap} \cdot \mathbf{m}\, d^3x \tag{2.26}$$

The energetic formulation allows to anticipate which field configurations $\mathbf{m}$ are the most benefited. A microstate of the universe specifies the state of the sample, $\mathbf{m}$, and the reservoir. The energy of the universe is constant hence for a given $\mathbf{m}$ there will be several reservoir configurations compatible with the total energy $e_{universe} = e_{sist} + e_{reserv}$. Hence for a given microstate for the system with energy, e_{sist}, the number of microstates of universe is proportional to the number of states of the reservoir, $\Omega'(e_{sist})$. It is known this quantity is proportional to $e^{-\beta e_{sist}}$. It is expected that if $e_{sist} = e_{ap}$ given above, then the more aligned is the magnetization with the applied field , the lower is e_{ap} and the higher will be the number of reservoir states, $\Omega'(e_{sist})$. This system configuration is therefore more probable when the system is in equilibrium. It is in this sence that the energy associated with a given field is said to contribute or penalise certain magnetization configurations of the system [13]. Despite the fact this argument is for externally applied fields, the same will work if the source of the field is other.

The dynamics described by the applied field can be understood discretizing time in LL equation giving us

$$\mathbf{m}' = \mathbf{m} - \mathbf{m} \times \mathbf{h}\, \Delta t - \alpha \mathbf{m} \times (\mathbf{m} \times \mathbf{h})\, \Delta t \tag{2.27}$$

where $\mathbf{m}' = \mathbf{m}(t + \Delta t)$ and $\mathbf{m} = \mathbf{m}(t)$ and $\mathbf{h} = \mathbf{h}_{ap}$. From the first term, at each time step it always updates the current state $\mathbf{m}$ with an increment $\mathbf{m} \times \mathbf{h}\, \Delta t$ that is orthogonal to $\mathbf{m}$. If $\Delta t \to 0$, this dynamics is represented by a circular trajectory around $\mathbf{h}_{ap}$.

The second term will add an increment that in turn is orthogonal to the first term, meaning it evolves $\mathbf{m}$ into pointing along the applied magnetic field. Observe that as this alignment occurs $|\mathbf{m} \times \mathbf{h}|$ gets smaller since $|\mathbf{m}| = 1$ and the applied field is constant. Therefore the

precession motion becomes gradually slower until it ceases when magnetization and the field are parallel.

The dynamics associated with $\mathbf{h}_{ap}$ describes for many steps, the process that occurs at one step when the field on the sample is not the applied field, $\mathbf{h}_{ap}$, but a more complicated field that at each step is changed, it can change because it will depend on the current system configuration, $\mathbf{m}(\mathbf{x}, t)$. These are the next contributors to be analysed.

Contribution 2: Uniaxial anisotropy

The lattice structure of a magnetic material benefits some spatial directions for the magnetic moments to align. A rigorous description would require quantum mechanics, but a simple formulation can be made the following way. Suppose that $\mathbf{m}$ belongs to a magnetic domain whose lattice has $\hat{e}_y$ as a favoured orientation. To describe this phenomenon it is introduced a magnetic field in that direction given that as in, $\mathbf{h}_{ap}$, a field in that direction will make $\mathbf{m}$ evolve until it coincides with $\hat{e}_y$. However, unlike an external applied field, a favoured direction only requires that $\mathbf{m}$ evolves until it coincides with the direction of $\hat{e}_y$, not its sense. To describe this, it is introduced on this $\hat{e}_y$ field a coefficient that depends on magnetization, $\kappa^{ad}(\hat{e}_z \cdot \mathbf{m})$, where the adimensional constant for uniaxial anisotropy, κ^{ad}, is given by $\frac{2\kappa}{\mu_0 M_s^2}$ [11, p. 33]. When positive, $\mathbf{m}$ evolves until superposition with $\hat{e}_y$, just like an applied field, $\mathbf{h}_{ap}$, in this direction, but if $(\hat{e}_z \cdot \mathbf{m})$ is negative, $\mathbf{m}$ evolves until it coincides with $-\hat{e}_z$.

For a generic favoured direction, this field is then

$$\mathbf{h}_{AN} = \hat{e}_{AN}\kappa^{ad}(\hat{e}_{AN} \cdot \mathbf{m}) \tag{2.28}$$

The energy formulation of such field is given by

$$e_{AN} = \int_V f_{AN}(\mathbf{m})\, d^3x \tag{2.29}$$

$$f_{AN}(\mathbf{m}) = \frac{\kappa^{ad}}{2}(1 - (\mathbf{m} \cdot \hat{e}_{AN})^2) \tag{2.30}$$

Observe that the energy is minimum when $\mathbf{m}$ coincides with $\hat{e}_{AN}$, that is $(\mathbf{m} \cdot \hat{e}_{AN})^2 = 1$. These are the most benefited configurations. To show the definition of f_{AN} leads to (2.28) we can recover the anisotropic field from the energy. Through variational calculus, suppose we have the first order variation in $\mathbf{m}$ is given by

$$\begin{aligned} \delta e_{AN} &= \frac{\kappa^{ad}}{2}\int (1 - ((\mathbf{m} + \delta\mathbf{m}) \cdot \hat{e}_{AN})^2) d^3x - \frac{\kappa^{ad}}{2}\int (1 - (\mathbf{m} \cdot \hat{e}_{AN})^2) d^3x \\ &= \frac{\kappa^{ad}}{2}\int (\delta\mathbf{m} \cdot \hat{e}_{AN})^2 d^3x - \kappa^{ad}\int (\mathbf{m} \cdot \hat{e}_{AN})(\delta\mathbf{m} \cdot \hat{e}_{AN}) d^3x \end{aligned} \tag{2.31}$$

The squared variation makes the first integral much smaller than the second, reformulating

the latter we have

$$\begin{aligned}\delta e_{AN} &= -\kappa^{ad}\int[(\mathbf{m}\cdot\hat{e}_{AN})\hat{e}_{AN}]\cdot\delta\mathbf{m}\,d^3x \\ &= -\kappa^{ad}\int\mathbf{h}_{AN}\cdot\delta\mathbf{m}\,d^3x\end{aligned} \tag{2.32}$$

where we identify the anisotropy field (2.28).

Contribution 3: The magnetic field of magnetization

The demagnetizing field, $\mathbf{h}_M$, can have a potential formulation satisfying a Poisson's equation under appropriate boundary conditions which will be the subject of next chapters. For now, its explicit solution is given by

$$\mathbf{h}_M = \frac{1}{4\pi}\int d^3x'\,\rho^*(\mathbf{x}')\frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|^3} + \frac{1}{4\pi}\int dS'\,\sigma^*(\mathbf{x}')\frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|^3} \tag{2.33}$$

where $\rho^* = -\nabla\cdot\mathbf{m}$ and $\sigma^* = \mathbf{m}\cdot\hat{n}$ are interpreted respectively as the density of fictitious magnetic charges within the system and on its boundary in an analogy with the electric field from polarized dielectric materials [3, p. 416].

The sum of energies associated to each magnetic moment in the presence of the field produced by all others is given by [3, p. 444]

$$e_M = -\frac{1}{2}\int d^3x\,\mathbf{h}_M\cdot\mathbf{m} = \frac{1}{2}\int d^3x\,|\mathbf{h}_M|^2 \tag{2.34}$$

where its first variation is obtained using the *reciprocity relation* [3, p. 387]

$$\begin{aligned}\delta e_M &= -\frac{1}{2}\int d^3x\,\delta\mathbf{h}_M\cdot\mathbf{m} - \frac{1}{2}\int d^3x\,\mathbf{h}_M\cdot\delta\mathbf{m} \\ &= -\int d^3x\,\mathbf{h}_M\cdot\delta\mathbf{m}\end{aligned} \tag{2.35}$$

The energy expression (2.34) allow us to know what equilibrium configurations are expected from LL update only with the demagnetizing field and to explain the second picture in figure 1.2. As shown, if we assume uniform magnetization along the y direction then within the system $\rho^* = 0$. On the bottom of the system we will have negative density of charges, $\sigma^* = (m_y\hat{e}_y)\cdot(-\hat{e}_y) < 0$, while on the left and right sides $\sigma^* = 0$ and on the top we have positive density. As a consequence the demagnetizing field will point downward, opposing the magnetization. Since the field is nonzero, from (2.34) the energy will be positive, $e_M > 0$. We conclude this configuration is not an energy minimum.

To be a minimum it is required to have no density of magnetic charges. Configurations as $\mathbf{m}$ circling the system tangent to its boundary give us $\rho^* = 0$ and $\sigma^* = 0$, the demagnetizing field is then $\mathbf{h}_M = 0$ and the energy is $e_M = 0$. However, experimentally it is found in ferromagnets the formation of domains of aligned magnetic moments represented as an example in figure 1.2. The presence of quantum exchange interactions allows its formation. A new energy and magnetic field term needs to be considered, the exchange field.

Contribution 4: The Exchange Field

The quantum exchange interaction [1] between closest neighborhoods magnetic moments can be represented by a local field derived from its energy formulation. From its quantum version [14, 15] it can derived the following equation:

$$e_{EX} = \frac{1}{2} \int_V \left((\nabla m_x)^2 + (\nabla m_y)^2 + (\nabla m_z)^2 \right) d^3x \tag{2.36}$$

where magnetization and distances are dimensionless, and we note that the exchange constant A that determines the strength of this contribution is contained in the definition of l_{EX} we use as a length scale (*cf* p. 9).

The first variation of the energy with the magnetization is given by

$$\delta e_{EX} = \frac{1}{2} \int \sum_{i=0}^{2} (\nabla(m_i + \delta m_i))^2 - (\nabla m_i)^2 \, d^3x \tag{2.37}$$

$$= \frac{1}{2} \int \sum_{i=0}^{2} 2\nabla m_i \cdot \nabla \delta m_i \, d^3x \tag{2.38}$$

where we expanded the squares and cancelled terms. Integrating by parts gives us

$$\delta e_{EX} = -\int \sum_{i=0}^{2} \delta m_i \nabla^2 m_i + \int \sum_{i=0}^{2} \delta m_i \nabla m_i \cdot \hat{n} \, dS \tag{2.39}$$

$$= -\int \delta \mathbf{m} \cdot \nabla^2 \mathbf{m} \, d^3x + \int ((\hat{n} \cdot \nabla)\mathbf{m}) \cdot \delta \mathbf{m} \, dS$$

If this variation is close to equilibrium, as we shall see in next section, we will have $(\hat{n} \cdot \nabla)\mathbf{m}$ close to zero, then we can identify the exchange magnetic field as given by

$$\mathbf{h}_{EX} = \nabla^2 \mathbf{m} \tag{2.40}$$

The functional form of the energy and field has a second derivative of magnetization, so the smother is the function $\mathbf{m}$, the smaller are the derivatives and so is the field and energy. These smooth configurations are those which have a minimum of energy. Given that LL updates the magnetization in order to minimize energy, it is to expect that from a initial high energy state, characterized by a spatially irregular function $\mathbf{m}$, that it evolve into a state where second derivatives are zero.

The combination of demagnetizing field and exchange field yields the formation of magnetic domains. The magnetization is aligned making the second derivatives zero and $\rho^* = 0$. In order to cancel fictitious magnetic charge density, σ^*, at the boundary of the domains, these are arranged for example as in the first picture in figure 1.2.

2.3 Equilibrium conditions

From previous section, we identified the equilibrium configurations from minimums of the energy expressions, these configurations are those that will not evolve in time, $\frac{\partial \mathbf{m}}{\partial t} = 0$, by LL or LLG equations. From the variational point of view, these minimums are obtained from the first variation in the energies. By collecting terms we want the magnetization such that for any variation, $\delta\mathbf{m}$, the total energy will not vary, that is

$$0 = \delta e_{tot} = -\int_V \left(\mathbf{h}_{ap} + \mathbf{h}_{AN} + \mathbf{h}_M + \nabla^2\mathbf{m}\right) \cdot \delta\mathbf{m}\, d^3x + \int_{\partial V} (\hat{n} \cdot \nabla)\mathbf{m} \cdot \delta\mathbf{m}\, dS \tag{2.41}$$

The variation needs to conserve the norm, that is $|\mathbf{m} + \delta\mathbf{m}| = 1$. There are two ways to implement this statement, using the method of Lagrange Multipliers or instead simply introduce the statement directly into the variation [5]. By defining a small rotation about $\delta\boldsymbol{\theta}$ with $\delta\mathbf{m} = \delta\boldsymbol{\theta} \times \mathbf{m}$, the equation (2.41) becomes

$$0 = -\int_V \delta\boldsymbol{\theta} \cdot (\mathbf{m} \times \mathbf{h}_{tot}) d^3x + \int_{\partial V} \delta\boldsymbol{\theta} \cdot (\mathbf{m} \times ((\hat{n} \cdot \nabla)\mathbf{m}) dS \tag{2.42}$$

where

$$\mathbf{h}_{tot} = \mathbf{h}_{ap} + \mathbf{h}_{AN} + \mathbf{h}_M + \nabla^2\mathbf{m} \tag{2.43}$$

Since we require this condition for any $\delta\boldsymbol{\theta}$, the terms in parenthesis in (2.42) need to be zero

$$\mathbf{m} \times \mathbf{h}_{tot} = 0 \tag{2.44}$$

$$\mathbf{m} \times ((\hat{n} \cdot \nabla)\mathbf{m}) = 0 \tag{2.45}$$

These are the Brown's conditions for equilibrium magnetization, the stability would require knowing the second variation of the energy [5]. The first condition implies that magnetization need to be aligned with the magnetic field in a equilibrium state, as expected experimentally. The second condition is

$$\mathbf{m} \times (\hat{n} \cdot \nabla)\mathbf{m} = 0 \tag{2.46}$$

which can be simplified. Noting either $\mathbf{m}$ is zero, or $(\hat{n} \cdot \nabla)\mathbf{m}$ is zero, or they are parallel. The first is excluded because the norm is nonzero, the last is excluded because norm cannot change, so the second must be true

$$(\hat{n} \cdot \nabla)\mathbf{m} = 0 \tag{2.47}$$

This condition will be used in next Chapter when second derivatives for exchange fields need to be computed near the boundary of the system.

2.4 Landau-Lishitz-Gilbert equation and Rayleigh dissipation

It is known from classical mechanics that friction and other dissipative forces can be approximated by the gradient of quadratic function in the velocities, the Rayleigh function [18]. Analogously, in micromagnetism we can introduce

$$R = a_0 \left(\frac{\partial m_x}{\partial t}\right)^2 + a_0 \left(\frac{\partial m_y}{\partial t}\right)^2 + a_0 \left(\frac{\partial m_z}{\partial t}\right)^2 \tag{2.48}$$

If the coefficients a_i are positive then the function will represent a bowl. For a given rate of change, $\dot{\mathbf{m}}$, the negative of the gradient will point in the opposite direction.

Remembering that in $\mathbf{m}$-space the magnetic field $\mathbf{h}$ determines the direction of evolution of magnetization $\mathbf{m}$, we can introduce the dissipative force as a magnetic field. To conserve the norm, the update must be orthogonal to $\mathbf{m}$, hence for the rotational term we have $-\mathbf{m}\times\mathbf{h}$. To introduce dissipation on this term we can add a new magnetic field, $\mathbf{h}_{dis}$, that opposes $\mathbf{h}$. Thus we define the following equation

$$\frac{\partial \mathbf{m}}{\partial t} = -\mathbf{m} \times (\mathbf{h} + \mathbf{h}_{dis}) \tag{2.49}$$

where

$$\begin{aligned} \mathbf{h}_{dis} &= -\nabla_{\dot{\mathbf{m}}} R \\ &= -\left(a_0 \frac{\partial m_x}{\partial t}, a_1 \frac{\partial m_y}{\partial t}, a_2 \frac{\partial m_z}{\partial t}\right) \end{aligned} \tag{2.50}$$

We can simplify the expression by choosing the coefficients as $a_0 = a_1 = a_2 = \alpha$ thereby giving us the LLG equation

$$\frac{\partial \mathbf{m}}{\partial t} = -\mathbf{m} \times \mathbf{h} + \alpha \mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t} \tag{2.51}$$

where the α coefficient is not the same one as in LL equation [11].

From [16] we note that LL and LLG determine different magnetization dynamics, and only if both α are small these are equivalent. The LLG equation has a dissipation term, $\mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t}$, that affects both the rotation term and the damping dynamics. If we increase α both dynamics are suppressed which is what we intuitively expect. This does not happen in LL, since both terms are orthogonal, and α only affects the damping term. It is for this distinction of LLG from the LL equation that we choose to integrate the former in the next chapters.

Chapter 3

Landau-Lifshitz-Gilbert equation and Finite Element Method

Magnetization dynamics also satisfies LLG equation. In this chapter we aim to introduce Newton-Raphson (NR) method for LLG, a well known technique for non-linear equations, after we descretize time using the Implicit Mid-point Rule [19, 12, 10].

We review the Finite Element Method to discretize space and introduce the necessary equations to compute each magnetic field term. This method allows simulation of curved shapes better than Finite Differences.

In [10] the discretized residuals for LLG and Poisson's equation are derived using Backwards-Euler method for approximating Implicit Midpoint Rule and the space discretization is done with basis function focused on the vertices [21, p. 38] using vectorial notation. In this work instead we will use basis functions focused on each element since it allows a better bridge between the equation and the code that will be develop in later Chapters, we will use Implicit Mid-Point Rule to discretize time. Additionally we will prefer using tensor notation in our equations, the Chapter is self-contained.

3.1 Statement of the problem

The goal is to compute the time evolution of magnetization $\mathbf{m}(\mathbf{x}) = (m_x, m_y, m_z)^T$ from an initial configuration that satisfies the LLG equation with the applied field, the stray field and exchange field contributions but we will not use anisotropy as the first three are the ones used in the standard problem we will use to validate the code. The LLG equation holds at every $\mathbf{x} \in \mathbb{R}^3$

$$\frac{\partial \mathbf{m}}{\partial t} = -\mathbf{m} \times \mathbf{h}_{tot} + \alpha_c \mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t} \tag{3.1}$$

$$\mathbf{h}_{tot} = \mathbf{h}_{ap} + \mathbf{h}_M + \nabla^2 \mathbf{m} \tag{3.2}$$

$$\mathbf{h}_M = -\nabla \phi \tag{3.3}$$

While in previous chapter the stray field was computed directly from the magnetization and hence only the system was considered and not its surrounding, in this chapter we will choose a new strategy and use the magnetic potential that satisfies Poisson's equation for all

space, $\mathbb{R}^3$. We will split it into the volume of the system, where $|\mathbf{m}| \neq 0$, defined to be Ω and a finite surroundings, Ω^c. The following picture represents these subvolumes.

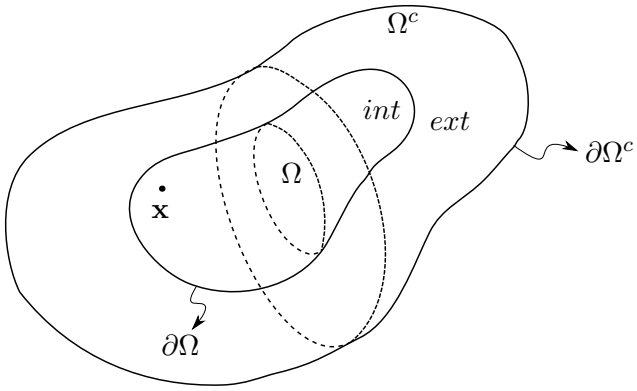

Figure 3.1: Representation of $d = 3$ subvolume Ω and a finite surroundings Ω^c, by extending the latter to infinity, $\Omega \cup \Omega^c = \mathbb{R}^3$. The dotted lines trace the outline of the surfaces of both subvolumes. The point $\mathbf{x}$ belongs to the surface $\partial\Omega$

From the theory of micromagnetism the demagnetizing field potential is given by

$$\nabla^2\phi = \begin{cases} \nabla \cdot \mathbf{m} & \text{if } \mathbf{x} \in \Omega \\ 0 & \text{if } \mathbf{x} \notin \Omega \end{cases} \tag{3.4}$$

Defining the two limits of a generic function $f(\mathbf{x})$ as $\mathbf{x}$ tends toward the boundary point $\mathbf{x}_0 \in \partial\Omega$ from inside (int) or outside (ext) as the values of the function $f_{int,ext}(\mathbf{x}_0)$

$$\begin{aligned} \lim_{\mathbf{x}\to\mathbf{x}_0\in\partial\Omega} f(\mathbf{x} \in \Omega) &= f_{int}(\mathbf{x}_0) \\ \lim_{\mathbf{x}\to\mathbf{x}_0\in\partial\Omega} f(\mathbf{x} \in \Omega^c) &= f_{ext}(\mathbf{x}_0) \end{aligned} \tag{3.5}$$

the matching conditions (MC) for the Poisson problem are

$$\begin{aligned} \frac{\partial\phi_{int}}{\partial n} - \frac{\partial\phi_{ext}}{\partial n} &= \mathbf{m} \cdot \hat{n} \\ \phi_{int} - \phi_{ext} &= 0 \end{aligned} \tag{3.6}$$

where $\mathbf{m} \cdot \hat{n}$ is evaluated at boundary value $\mathbf{x}_0$. Additionally we require the asymptotic behaviour $\phi \to 0$ as $|\mathbf{x}| \to \infty$. The presence of Poissons's equation and of a time derivative on the RHS of LLG is what makes this problem distinct from LL, it's a implicit problem and new techniques such as the Newton-Raphson method and Finite Element Method will be used, but first we will introduce a time discretization known as Implicit Mid-Point Rule.

3.2 Implicit Mid-Point Rule and Newton-Raphson Method

First we will write LLG showing its dependencies

$$\frac{\partial \mathbf{m}}{\partial t} = f\left(t, \mathbf{m}, \phi, \frac{\partial \mathbf{m}}{\partial t}\right) \tag{3.7}$$

where the explicit dependence on time comes from $\mathbf{h}_{ap}$. However, for our case, we assume the applied field is constant. Since time is a one dimensional variable, discretization of derivatives only requires replacement by finite differences

$$\frac{\partial \mathbf{m}}{\partial t} \longrightarrow \frac{\mathbf{m}(t+\Delta t) - \mathbf{m}(t)}{\Delta t} \tag{3.8}$$

The idea behind IMR is to compute f at the average magnetization and potential between the current configuration and the next one, its arguments have then the following substitutions

$$t \longrightarrow \frac{(t+\Delta t)+t}{2} = t_{mid} \tag{3.9}$$

$$\mathbf{m} \longrightarrow \frac{\mathbf{m}(t+\Delta t)+\mathbf{m}(t)}{2} = \mathbf{m}_{mid} \tag{3.10}$$

$$\phi \longrightarrow \frac{\phi(t+\Delta t)+\phi(t)}{2} = \phi_{mid} \tag{3.11}$$

Observe that f depends now on the magnetization at the next instant of time. Substituting into Poisson's equation as well we have for a given $\mathbf{m}(t)$ and associated $\phi(t)$ the following equations

$$\frac{\mathbf{m}(t+\Delta t) - \mathbf{m}(t)}{\Delta t} = f\left(t_{mid}, \mathbf{m}_{mid}, \phi_{mid}, \frac{\mathbf{m}(t+\Delta t) - \mathbf{m}(t)}{\Delta t}\right) \tag{3.12}$$

$$\nabla \phi_{mid} = \nabla \cdot \mathbf{m}_{mid} \tag{3.13}$$

that implicitly determine the updated configuration $\mathbf{m}(t+\Delta t)$ and $\phi(t+\Delta t)$.

The Newton-Raphson strategy of finding the magnetization $\mathbf{m}(t+\Delta t)$ and potential $\phi(t+\Delta t)$ that solve (3.12) and (3.13) is to build for each equation a relation such that these solutions are their fixed points, we name these exact solutions, $\mathbf{m}^*$ and ϕ^* [17]. We define the current configuration as $\mathbf{m}_0 = \mathbf{m}(t)$ and $\phi_0 = \phi(t)$ and a generic configuration at $t+\Delta t$ as $\mathbf{m} = \mathbf{m}(t+\Delta t)$ and $\phi = \phi(t+\Delta t)$ not necessarily satisfying these equations, whose residuals are

$$\mathbf{r}(\mathbf{m}, \phi) = \frac{\mathbf{m} - \mathbf{m}_0}{\Delta t} - f\left(t_{mid}, \frac{\mathbf{m}+\mathbf{m}_0}{2}, \frac{\phi+\phi_0}{2}, \frac{\mathbf{m}-\mathbf{m}_0}{\Delta t}\right) \tag{3.14}$$

$$s(\mathbf{m}, \phi) = \nabla \cdot \left(\frac{\mathbf{m}+\mathbf{m}_0}{2}\right) - \nabla^2\left(\frac{\phi+\phi_0}{2}\right) \tag{3.15}$$

Observe that while $\mathbf{m}$ and ϕ are variables, $\mathbf{m}_0$ and ϕ_0 work as parameters that we have to choose. For given ones, different residuals are obtained associated with the configurations, $\mathbf{m}$ and ϕ, that measure how close they are to LLG and Poisson's exact solutions. If we set up $\mathbf{r} = 0$ and $s = 0$ for all $\mathbf{x} \in \mathbb{R}^3$, these two equations determine $\mathbf{m}^*$ and ϕ^*, that are the

next step from $\mathbf{m}_0$ and ϕ_0 the LLG equation determines while simultaneously satisfying the Poisson's equation.

A practical approach to find $\mathbf{m}^*$ and ϕ^* is to choose $\mathbf{m}_0$ and ϕ_0 and then approximate (3.14) and (3.15) by a first order Taylor series around a particular configuration $\mathbf{m}_p$ and ϕ_p that we must also specify

$$\mathbf{r}(\mathbf{m},\phi) \approx \mathbf{r}_{cutoff}(\mathbf{m},\phi) = \mathbf{r}(\mathbf{m}_p,\phi_p) + \frac{\partial \mathbf{r}(\mathbf{m}_p,\phi_p)}{\partial \mathbf{m}} \cdot (\mathbf{m}-\mathbf{m}_p) + \frac{\partial \mathbf{r}(\mathbf{m}_p,\phi_p)}{\partial \phi}(\phi-\phi_p) \quad (3.16)$$

$$s(\mathbf{m},\phi) \approx s_{cutoff}(\mathbf{m},\phi) = s(\mathbf{m}_p,\phi_p) + \frac{\partial s(\mathbf{m}_p,\phi_p)}{\partial \mathbf{m}} \cdot (\mathbf{m}-\mathbf{m}_p) + \frac{\partial s(\mathbf{m}_p,\phi_p)}{\partial \phi}(\phi-\phi_p) \quad (3.17)$$

It's important to notice the derivatives are with respect to the variables $\mathbf{m}$ and ϕ and then evaluated at the point of expansion $\mathbf{m}_p$ and ϕ_p. From these equations an iterative method can be devised. We start by solving $\mathbf{r}_{cutoff}(\mathbf{m},\phi) = 0$ and $s_{cutoff}(\mathbf{m},\phi) = 0$ with $\mathbf{m}_p = \mathbf{m}_0$ and $\phi_p = \phi_0$, if the solution $\mathbf{m}$ and ϕ are closer to $\mathbf{m}^*$ and ϕ^*, we can repeat the process using these as the new $\mathbf{m}_p$ and ϕ_p, progressively evolving $\mathbf{m}$ toward $\mathbf{m}^*$ and ϕ to ϕ^*, but once the residual is small enough we can stop the process, yielding two approximations, $\mathbf{m}'$ and ϕ', to the exact solutions. We then reparametrize the linear approximations with $\mathbf{m}_0 = \mathbf{m}'$ and $\phi_0 = \phi'$, and refresh the process looking for the new step in the evolution of magnetization and potential.

The solutions we will search for (3.12) and (3.13) are not $\mathbf{m}^*$ and ϕ^* but approximated ones, $\mathbf{m}^*_{\approx}$ and $\phi^*_{\approx}$, since it will not be the residuals (3.16) and (3.17) we will use. Instead we will start by putting both in what we will later call as their weak forms and then introduce a spacial discretization. Its these discretized residuals that we will linearly approximate. For that we will require expressions for the discretized $\mathbf{r}(\mathbf{m},\phi)$ and $s(\mathbf{m},\phi)$ and then compute their Jacobians yielding explicitly a 3×3 matrix for $\frac{\partial \mathbf{r}}{\partial \mathbf{m}}$, a 1×3 row vector for $\frac{\partial s}{\partial \mathbf{m}}$, a 3×1 column vector for $\frac{\partial \mathbf{r}}{\partial \phi}$ and a scalar for $\frac{\partial s}{\partial \phi}$. In the next section we introduce the well known Finite Element Method and expressions for the entries for both residuals and Jacobians.

3.3 Domain discretization and basis functions

The spatial domain where the potential function is defined can be tesselated into tetrahedra, characterized by four vertices connected by edges. As an example take the cube in figure 3.2, which has 8 vertices plus a central one. We can divide it into 12 tetrahedra of which one is shown, like all the others, it has the vertex 8 and the remaining three on one of the facets of the cube, vertices 4, 5 and 6 for this particular element. Observe these tetrahaedra fill completely the cube leaving no holes, and all edges connecting the vertices never cross, these properties characterize tetrahedral meshes for generic domains, observe also that the 12 tetrahedra enclose the vertex 8 and the inner triangular facets are shared by two tetrahedra while the surface ones do not.

Building meshes for arbitrary shapes as in figure 3.1 will require these two main properties, the shapes of tetrahedra might be different but ideally each element should have all six edges with the same length as this minimizes the error in FEM, not the case for our BCC example where $\overline{56} > \overline{68} = \overline{48} = \overline{58} > \overline{46} = \overline{45}$.

Table 3.1: Connectivity matrix for figure 3.2. Each column represents a tetrahedral element and each row all vertices with local index α.

(α, e)	0	1	2	3	4	5	6	7	8	9	10	11
0	0	2	0	0	0	0	2	2	5	4	1	3
1	2	4	4	1	2	1	6	3	6	5	3	5
2	4	6	5	5	3	3	7	7	7	6	5	7
3	8	8	8	8	8	8	8	8	8	8	8	8

In FEM [17] a mesh structure for $\Omega \cup \Omega^c$ allows one to replace the problem of finding an exact solution for the Poisson equation, $\phi^*(\mathbf{x})$, by the finding of an approximated one, $\phi_h(\mathbf{x})$, where the index h indicates that this is an approximated version of the exact solution within the tetrahedron element of index e. The approximation consists in considering the behaviour of ϕ_h up to a given order, we will assume just a linear behaviour within each tetrahedrum, it will depend only on the potential on the four vertices, so we only have to find a finite number of values for ϕ_h, the n_q vertices of the mesh.

The Poisson's equation which determines locally the behaviour of the exact solution is reformulated into its weak form to be introduced later, so that instead it determines the behaviour of ϕ_h by relating neighbouring vertices: for each vertex a linear equation is associated, the set of these equations determine a linear system of equations whose solution is the potential at each vertex.

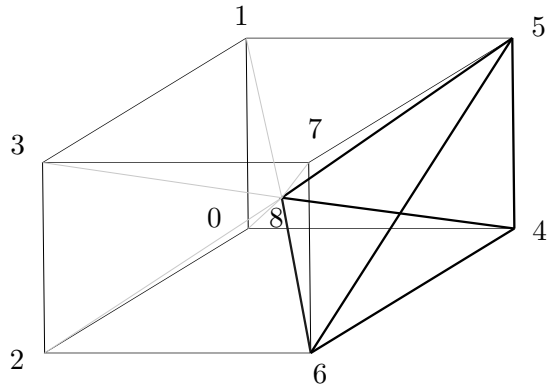

Figure 3.2: A body-centered cube (BCC) with 9 vertices and 12 tetrahedral elements. Cube's facet edges not drawn except for tetrahedron 4568.

To build a system of equations that relate each vertex with its neighbours connected by edges, we need to represent the mesh by identifying each vertex and describe how they are connected. We define then the connectivity matrix for tetrahedra, a matrix with four rows and a number of columns equal to the number of tetrahedra, n_e.

First we index all vertices $i \in \{0, 1, 2, ..., n_q - 1\}$, in any order, just as in figure 3.2, and then the tetrahedron shaped elements with index $e \in \{0, 1, ..., n_e - 1\}$, which although not shown in the picture to simplify it, belong to $\{0, 1, 2, ..., 11\}$. Then for each element, e, the associated four vertices are numbered again in any order, but by local indices, $\alpha \in \{0, 1, 2, 3\}$ corresponding to the four rows of the matrix at the column e. The matrix entry (e, α) identifies uniquely with local variables one vertex of the mesh, the vertex we assigned the glocal index i. These are the entries in table 3.1, where we have the tetrahedron in figure 3.2

Table 3.2: Connectivity matrix for surface elements in figure 3.2. Each column represents triangle element and each row all vertices with local index α. The α indices need not coincide with table 3.1.

(α, s)	0	1	2	3	4	5	6	7	8	9	10	11
1	2	4	4	1	2	1	6	3	6	5	3	5
0	0	2	0	0	0	0	2	2	5	4	1	3
2	4	6	5	5	3	3	7	7	7	6	5	7

with index $e = 9$ since all its four vertices are in the tenth column. All other tetrahedra not shown in figure 3.2 are represented as all other columns.

Another connectivity matrix for the surface vertices can be built. Keeping the same global indices for the vertices, we index the n_s surface triangles in any order, $s \in \{0, 1, 2, ..., n_s - 1\}$. In this simple example in figure 3.2, the boundary connectivity matrix in table 3.2 has as many tetrahedral elements as there are surface elements, also the first three rows of table 3.1 coincide with table 3.2, generally this is not the case and $n_e \gg n_s$.

Once built, the connectivity matrix will allow us to convert from indices (e, α) into i or for example to identify easily all vertices with $\alpha = 2$, as they all are in the third row. It works like a function, we will name it $i = n(e, \alpha)$ or $i = n(s, \alpha)$ depending on whether we are referring to tetrahedral elements or triangular elements. With them we can define sets like $\{(e, \alpha)|i = n(e, \alpha)\}$ which identifies all vertices, in (e, α) notation that correspond to the same vertex of the mesh and also gives us the elements they are in, this is important for the equations we derive next since calculations associated with a vertex i can be split into calculations on each tetrahedra that shares this vertex.

Now that the mesh structure is well defined we can identify the four vertices of an element e by the four global indices in column e, we then easily pick their locations by preconstructing an array with three columns, for x, y and z and where each row is assigned i. With these four vertices locations we can obtain the linear representation of the potential or any other function of space within the tetrahedra. We proceed as follows for the potential case, define it within the tetrahedra e as $\phi_h(\mathbf{x})$ with $\mathbf{x} \in \Omega^e$ and suppose it can be written as

$$\phi_h(\mathbf{x}) = \begin{cases} a^e + b^e x + c^e y + d^e z & \text{if } \mathbf{x} \in \Omega^e \\ 0 & \text{if } \mathbf{x} \notin \Omega^e \end{cases} \tag{3.18}$$

where a^e, b^e, c^e and d^e are unknowns to be determined that characterize the function inside the element. Close neighbour elements of a given vertex i, $\{(e, \alpha)|i = n(e, \alpha)\}$, need to output the same value for $\phi(\mathbf{x}_i) = \phi_i$ to give us a continuous solution. To sustain this consistency through out the mesh we need the values of ϕ_i to determine these coefficients, in local coordinates the four potential for a single tetrahedron are ϕ_0^e, ϕ_1^e, ϕ_2^e and ϕ_3^e, the following system of equations allow us to relate these with the coefficients

$$\begin{pmatrix} 1 & x_0^e & y_0^e & z_0^e \\ 1 & x_1^e & y_1^e & z_1^e \\ 1 & x_2^e & y_2^e & z_2^e \\ 1 & x_3^e & y_3^e & z_3^e \end{pmatrix} \begin{pmatrix} a^e \\ b^e \\ c^e \\ d^e \end{pmatrix} = \begin{pmatrix} \phi_0^e \\ \phi_1^e \\ \phi_2^e \\ \phi_3^e \end{pmatrix} \tag{3.19}$$

where we define the matrix as X.

By inverting the system of equations each of the coefficients is determined by the vertices potentials, ϕ^e_α. With the coefficients the potential inside the element, e, comes determined as well when we use (3.18), that is, the interior information is linearly interpolated from the boundary vertices' potentials. We can invert the system of equations by introducing the cofactors of matrix X as

$$\begin{pmatrix} det(X) & & & \\ & det(X) & & \\ & & det(X) & \\ & & & det(X) \end{pmatrix} = \begin{pmatrix} 1 & x^e_0 & y^e_0 & z^e_0 \\ 1 & x^e_1 & y^e_1 & z^e_1 \\ 1 & x^e_2 & y^e_2 & z^e_2 \\ 1 & x^e_3 & y^e_3 & z^e_3 \end{pmatrix} \begin{pmatrix} c_{00} & c_{10} & c_{02} & c_{30} \\ c_{01} & c_{11} & c_{21} & c_{31} \\ c_{02} & c_{12} & c_{22} & c_{32} \\ c_{03} & c_{13} & c_{23} & c_{33} \end{pmatrix} \tag{3.20}$$

Then the coefficients are

$$\begin{pmatrix} a^e \\ b^e \\ c^e \\ d^e \end{pmatrix} = \frac{1}{det(X)} \begin{pmatrix} c_{00} & c_{10} & c_{02} & c_{30} \\ c_{01} & c_{11} & c_{21} & c_{31} \\ c_{02} & c_{12} & c_{22} & c_{32} \\ c_{03} & c_{13} & c_{23} & c_{33} \end{pmatrix} \begin{pmatrix} \phi^e_0 \\ \phi^e_1 \\ \phi^e_2 \\ \phi^e_3 \end{pmatrix} \tag{3.21}$$

Substituting these coefficients in (3.18) we get

$$\begin{aligned} \phi_h(\mathbf{x}) = \frac{1}{det(X)}[&(c_{00} + c_{10}x + c_{20}y + c_{30}z)\phi^e_0 + \\ &(c_{10} + c_{11}x + c_{12}y + c_{13}z)\phi^e_1 + \\ &(c_{20} + c_{12}x + c_{22}y + c_{23}z)\phi^e_2 + \\ &(c_{30} + c_{13}x + c_{32}y + c_{33}z)\phi^e_3] \end{aligned} \tag{3.22}$$

Observe that $\phi_h(\mathbf{x})$ turns out to be a linear combination of linear functions, whose coefficients are the cofactors of the matrix X. These are the basis function for the potential in element e, they are important from now on, so we name them

$$N^e_\alpha(\mathbf{x}) = \frac{1}{det(X)}(c^e_{\alpha,0} + c^e_{\alpha,1}x + c^e_{\alpha,2}y + c^e_{\alpha,3}z) \tag{3.23}$$

and observe that $det(X) = 6\bar{\Omega}^e$ where $\bar{\Omega}^e$ is the signed volume [21], that is the volume which can be positive or negative depending on how the rows of X were organized which in turn depends on how we numbered the vertices locally. By having the signed volume, the order of numbering is irrelevant.

With the shape functions we can approximate within element e the potential as

$$\phi(\mathbf{x}) \approx \phi_h(\mathbf{x}) = \sum_{\alpha=0}^{3} N^e_\alpha(\mathbf{x})\phi_\alpha \tag{3.24}$$

where its only nonzero within Ω^e. These linear basis functions are also known as shape functions and have an important property, notice the factor in parenthesis in (3.23). It has almost the $det(X)$ form. The α index indicate the row of X, where $(x^e_\alpha, y^e_\alpha, z^e_\alpha) = \mathbf{x}^e_\alpha$ was

replace by $\mathbf{x}$. If we pick $\mathbf{x} = \mathbf{x}_\alpha^e$ then $N_\alpha^e(\mathbf{x}_\alpha^e) = 1$ in (3.23). But if $\mathbf{x}$ is any other of the three tetrahedron vertices locations or any linear combination of these, then the basis function is zero. Basis functions with these properties allow $\phi^e(\mathbf{x}_\alpha^e) = \phi_\alpha^e$ as required. We conclude

$$N_\alpha^e(\mathbf{x}_\beta^e) = \delta_{\alpha\beta} \tag{3.25}$$

Not only we can expand scalar function on this basis, also vector functions like magnetization can be expanded as follows

$$\mathbf{m}_h = \sum_{\alpha=0}^{3} \mathbf{m}_\alpha^e N_\alpha^e(\mathbf{x}) \tag{3.26}$$

We can now extend the argument and represent the functions for all tetrahedra by summing over all elements as

$$\phi_h(\mathbf{x}) = \sum_{e=0}^{n_e-1} \sum_{\alpha=0}^{3} \phi_\alpha^e N_\alpha^e(\mathbf{x}) \tag{3.27}$$

$$\mathbf{m}_h(\mathbf{x}) = \sum_{e=0}^{n_e-1} \sum_{\alpha=0}^{3} \mathbf{m}_\alpha^e N_\alpha^e(\mathbf{x}) \tag{3.28}$$

However care must be taken when we evaluate at a given $\mathbf{x}_i$ (it will be useful in section 4.3) since all basis function *incident* on i, $\{N_\alpha^e(\mathbf{x}) | i = n(e,\alpha)\}$ yield the value of 1. Defining the number of these functions as $N_{inc} = \#\{(e,\alpha) | i = n(e,\alpha)\}$, the potential will be $\phi_h(\mathbf{x}_i) = N_{inc} \times \phi_i$. To prevent this N_{inc} factor when we use either (3.27) or (3.28), we will assume that the domain for each basis function incident on i does not superimpose and only one of the basis functions is actually 1 on $\mathbf{x}_i$. Therefore we have $\phi_h(\mathbf{x}_i) = \phi_i$ and $\mathbf{m}_h(\mathbf{x}_i) = \mathbf{m}_i$. Notice this assumption is irrelevant when computing integrals on $\mathbb{R}^3$ that involve (3.27) or (3.28).

Finally the basis has the following important property as well, for a given tetrahedral element the basis is not orthogonal, instead satisfies the following relation [21]

$$\int_{\Omega^e} (N_0^e)^k (N_1^e)^l (N_2^e)^m (N_3^e)^n d^3x = \frac{k!l!m!n!}{(k+l+m+n+3)!} 6\Omega^e \tag{3.29}$$

for surface triangular elements N_α^e has the same values that a triangular basis function would, the latter satisfies

$$\int_{\Omega^s} (N_0^s)^k (N_1^s)^l (N_2^s)^m dS = \frac{k!l!m!}{(k+l+m+2)!} 2\Delta^s \tag{3.30}$$

Both will be useful when evaluating residual integrals in next sections.

3.4 Weak form of Poisson's residual

Since within the element e, ϕ_h is a linear function, when we substitute into (3.4) its Laplacian is zero. Poisson's equation is a strong statement since it excludes solutions of the form (3.27), but multiplying by a basis function, integrating we get

$$\int_{\Omega^e} \nabla^2 \phi N_\alpha^e(\mathbf{x}) d^3x = \int_{\Omega^e} \nabla \cdot \mathbf{m} N_\alpha^e(\mathbf{x}) d^3x \tag{3.31}$$

and while before we had the Poisson's equation (3.4) relating the local curvature of the function ϕ with the divergence of the magnetization and this determine the solution, with this integral equation (3.31) we have on its LHS the sum of second derivatives weighted by the shape function, equal to the RHS where we have the sum of divergences of magnetization also weighted by the shape function on element Ω^e. This statement is less restrictive because takes into account information from the entire element, it determines a larger set of ϕ solution for (3.31) [10].

However to allow a linear solution we still have to go further. Integrating by parts, we can transfer one of the gradients to $N_\alpha^e(\mathbf{x})$ at the price of a minus sign and a boundary term, rearranging we define the Poisson's residual in its weak formulation

$$s_{\phi,\alpha}^e = \int_{\Omega^e} \nabla \cdot \mathbf{m} N_\alpha^e(\mathbf{x}) d^3x + \int_{\Omega^e} \nabla\phi \cdot \nabla N_\alpha^e(\mathbf{x}) d^3x - \int_{\partial\Omega^e} N_\alpha^e(\mathbf{x}) \nabla\phi \cdot \hat{n} dS \tag{3.32}$$

When we set it equal to zero we obtain the weak form of Poisson's equation. However our goal is to implement Newton-Raphson, hence its the residual that we will work with. We can transform it into a set of linear expressions by substituting ϕ_h and $\mathbf{m}_h$, this is possible since the second integral now isn't zero, instead of the Laplacian we have the gradient of the basis function which is given by

$$\nabla N_\alpha^e(\mathbf{x}) = \frac{1}{det(X)} \begin{pmatrix} c_{1,\alpha} \\ c_{2,\alpha} \\ c_{3,\alpha} \end{pmatrix} \tag{3.33}$$

From (3.32), the residual at vertex i can now be computed, summing contributions from elements that share this vertex we have

$$s_i = \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} s_\alpha^e \tag{3.34}$$

Notice that before discretization we had for every function $\phi(\mathbf{x})$ we substitute in (3.15) a residual value for every $\mathbf{x} \in \mathbb{R}^3$, now after discretization we have it for each vertex of the mesh, s_i. Arranging it into a vector we get

$$\mathbf{s} = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n_q-1} \end{pmatrix} \tag{3.35}$$

which is the discreet version of the continuous function $s(\mathbf{x})$ in (3.32), information about its behaviour is lost but now we only need to handle an n_q entry vector, which can better approximate it's continuous counterpart if we reduce the element size and thereby increase the number of n_q of entries in (3.35).

Each one of the three integrals in (3.32) is a contribution for the $i = n(e, \alpha)$ entry of a column vector, they behave differently depending of the the location of vertex, $\mathbf{x}_i$. We will analyse them starting with the source term (src), followed by the stiffness term (stf) and finally the surface integral (∂) where Neumann boundary conditions and matching conditions enter. The sum total is given by

$$s_{\phi,i} = s^{src}_{\phi,i} + s^{stf}_{\phi,i} - s^{\partial}_{\phi,i} \tag{3.36}$$

However, if we suppose we have instead a Poisson's equation with Dirishlet boundary conditions on the boundary of the system, $\partial\Omega$, then there would not be a surface integral and residual contribution $s^{\partial}_{\phi,i}$ to take into account [21, p. 92, 108].

1. The Source term

We start by approximating the divergence using the discretized magnetization

$$\nabla \cdot \mathbf{m} \approx \nabla \cdot \mathbf{m}_h = \sum_{\alpha=0}^{3} \mathbf{m}^e_\alpha \cdot \nabla N^e_\alpha \tag{3.37}$$

the source term integral in (3.36) is given by

$$s^{e,src}_{\phi,\alpha} = \int_{\Omega^e} \nabla \cdot \mathbf{m}_h N^e_\beta(\mathbf{x}) d^3x = \frac{\Omega^e}{4} \sum_{\beta=0}^{3} \mathbf{m}^e_\beta \cdot \nabla N^e_\beta \tag{3.38}$$

where we used the fact $\int_{\Omega^e} N^e_\alpha(\mathbf{x}) d^3x = \frac{\Omega^e}{4}$ from (3.29). This contribution only depend on e. To sum all contributions for the vertex i is to sum factors for each e that have i on its vertices.

$$s^{src}_{\phi,i} = \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} s^{e,src}_{\phi,\alpha} = \sum_{\{e|\exists\alpha(i=n(e,\alpha))\}} \sum_{\beta=0}^{3} \frac{\Omega^e}{4} \mathbf{m}^e_\beta \cdot \nabla N^e_\beta \quad \text{if} \quad i \in \Omega \cup \Omega^c \tag{3.39}$$

Observe that for $i \in \Omega \setminus \partial\Omega$ all terms of the double sum are nonzero since this is completely within the magnetized material. If $i \in \partial\Omega$ then terms associated with elements outside the magnetized region will be zero. For i within the vacuum region, then the source residual term is zero, we have a Laplace's equation.

Substituting now IMR, we have

$$\text{with IMR: } s^{src}_{\phi,i} = \sum_{\{e|\exists\alpha(i=n(e,\alpha))\}} \sum_{\beta=0}^{3} \frac{\Omega^e}{8} (\mathbf{m}^e_\beta + \mathbf{m}^e_{\beta,0}) \cdot \nabla N^e_\beta \tag{3.40}$$

There are n_q residuals like these, as many as the number of vertices in the magnetized region plus the surrounding vacuum, with them we can build a vector, $\mathbf{s}^{src}_\phi$, which will be summed to the other two integral terms we analyse next.

2. The Stiffness Term

Introducing the descretized potential from (3.24) into the second integral of (3.32) we get

$$s_{\phi,\alpha}^{e,stf} = \int_{\Omega^e} \nabla\phi_h \cdot \nabla N_\alpha^e(\mathbf{x}) d^3x = \Omega^e \sum_{\beta=0}^{3} \phi_\beta^e \nabla N_\beta^e \cdot \nabla N_\alpha^e \tag{3.41}$$

Summing now all contribution from all elements that have the vertex i we have

$$s_{\phi,i}^{stiff} = \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} s_{\phi,\alpha}^{e,stiff} = \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \sum_{\beta=0}^{3} \phi_\beta^e \nabla N_\beta^e \cdot \nabla N_\alpha^e \Omega^e \quad \text{if} \quad i \in \Omega \cup \Omega^c \tag{3.42}$$

$$\text{with IMR:} \quad s_{\phi,i}^{stiff} = \frac{1}{2} \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \sum_{\beta=0}^{3} (\phi_\beta^e + \phi_{\beta,0}^e) \nabla N_\beta^e \cdot \nabla N_\alpha^e \Omega^e \tag{3.43}$$

Many terms of this double summation in (3.42) will have ϕ_β^e corresponding to the same ϕ_j, where j is a global index of a vertex connected to i. Lumping all its coefficients into a single constant, we define it as $K_{i,j}$.

$$K_{ij} = \sum_{\{(e,\alpha,\beta)|i=n(e,\alpha)\wedge j=n(e,\beta)\}} \nabla N_\beta^e \cdot \nabla N_\alpha^e \Omega^e \tag{3.44}$$

We see then, $s_{\phi,i}^{stiff}$ as a sum of products of each $K_{i,j}$ with the corresponding ϕ_j. By going further we extend this sum by adding the multiplications of $K_{i,j} = 0$ with ϕ_j for all other j not connected to i. This shows that we can express $s_{\phi,i}^{stiff}$ as a dot product of a row vector with $K_{i,j}$ as entries with a column vector $\boldsymbol{\phi}$. Since all other indices i could have been chosen, we have for each one a dot product like this, all these calculations can be arranged in a column vector $\mathbf{s}_\phi^{stiff}$ which is then given by the product of the matrix K with the vector $\boldsymbol{\phi}$

$$\mathbf{s}_\phi^{stiff} = \frac{1}{2} K(\boldsymbol{\phi} + \boldsymbol{\phi}_0) \tag{3.45}$$

For a given row i of K the only nonzero entries are those whose columns that are associated to vertices connected to i including the diagonal entries that correspond to $\sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \nabla N_\alpha^e \cdot \nabla N_\alpha^e$. Since most of the entries of K will be zero, we designate K as a sparse matrix.

The $n_q \times 1$ vector $\boldsymbol{\phi}$ is multiplied by the $n_q \times n_q$ sparse matrix K to yield the second piece of $\mathbf{s}$ for (3.35).

3. The Surface term

The surface integration is made along the four triangle facets of a tetrahedron, $\partial\Omega^e$

$$s_{\phi,\alpha}^{e,\partial} = \int_{\partial\Omega^e} N_\alpha^e(\mathbf{x}) \nabla\phi_h \cdot \hat{n} dS \tag{3.46}$$

The surface contribution for a vertex i is obtained like the two previous integral terms by adding the contributions of all tetrahedra that have i as one of its vertices, we write

$$s^{\partial}_{\phi,i} = \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \int_{\partial\Omega^e} N^e_\alpha(\mathbf{x}) \nabla\phi \cdot \hat{n} dS \tag{3.47}$$

The result of such a calculation depends on where the vertex i is located. We have four cases to consider.

If $i \in \Omega \setminus \partial\Omega$ or $i \in \Omega^c$ the surface is composed of triangles that form a "shell-like" surface that is integrated once with $\hat{n}$ pointing outwards, analogous to $i = 8$ in figure 3.2. Remembering the basis functions are equal to 1 at the core of this shell, $\mathbf{x}_i$, all basis function will be zero on its surface. Integrations along triangular facets inside the shell will be made twice with $\hat{n}$ pointing in opposite directions, since $\nabla\phi \cdot \hat{n}$ is continuous within or outside Ω, these terms cancel in pairs. Hence $s^{\partial}_{\phi,i}$ will be zero within these two subvolumes. However if $i \in \partial\Omega$, some of these facets belong to $\partial\Omega$, as an example consider the following figure where

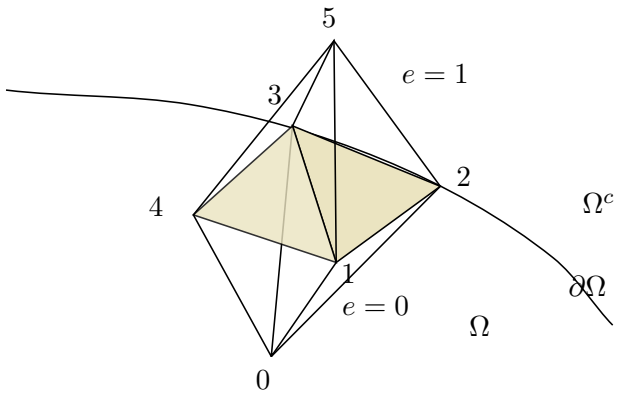

Figure 3.3: Four elements with shared facets belonging to the tessellated surface $\partial\Omega$ (shaded).

we represent only four of the elements that will enclose the central vertex $i = 1$ that belongs to the surface $\partial\Omega$. Two terms of the summation in (3.47) for this case are

$$s^{\partial}_{\phi,i=1} = \int_{\Delta_{123}} \nabla\phi_{int} \cdot \hat{n} N^{e=0}_\alpha(\mathbf{x}) dS + \int_{\Delta_{123}} \nabla\phi_{ext} \cdot (-\hat{n}) N^{e=1}_\alpha(\mathbf{x}) dS + ... \tag{3.48}$$

$$= \int_{\Delta_{123}} \mathbf{m} \cdot \hat{n} N^{e=0}_\alpha(\mathbf{x}) dS + ... \tag{3.49}$$

where $n(e = 0, \alpha) = n(e = 1, \beta) = 1$ hence both basis function for different elements will give the same output if evaluated on the same triangle, Δ_{123}, by (3.6) the derivatives on $\partial\Omega$ are discontinuous, therefore the integrals do not cancel. Generically we will have to sum all integrations along each surface triangle in $\partial\Omega$ that has i as one of its vertices, we then write the nonzero part of (3.46) as

$$s^{s,\partial}_{\phi,\alpha} = \int_{\Delta^s} \mathbf{m}_h \cdot \hat{n} N^s_\alpha(\mathbf{x}) dS \tag{3.50}$$

Summing for all contribution for vertex i

$$\begin{aligned} s^{s,\partial}_{\phi,\alpha} &= \int_{\Delta^s} \mathbf{m}_h \cdot \hat{n} N^s_\alpha(\mathbf{x}) dS \\ &= \sum_{\beta=0}^{2} \mathbf{m}^s_\beta \cdot \hat{n}^s \times \begin{cases} \frac{1}{12}\Delta^s & \text{if } \alpha \neq \beta \\ \frac{1}{6}\Delta^s & \text{if } \alpha = \beta \end{cases} \quad \text{if } n(s,\alpha) \in \partial\Omega \end{aligned} \tag{3.51}$$

$$\text{with IMR:} \quad s^{s,\partial}_{\phi,\alpha} = \sum_{\beta=0}^{2} (\mathbf{m}^s_\beta + \mathbf{m}^s_{\beta,0}) \cdot \hat{n}^s \times \begin{cases} \frac{1}{24}\Delta^s & \text{if } \alpha \neq \beta \\ \frac{1}{12}\Delta^s & \text{if } \alpha = \beta \end{cases} \tag{3.52}$$

where observe we substituted $\mathbf{m}_h = \sum_{\beta=0}^{2} \mathbf{m}^s_\beta N^s_\beta$, that is all (e,β) vertices also belong to $\partial\Omega$ and we used the integral (3.30).

The final case occurs when vertex i belongs to the vacuum domain surface $\partial\Omega^c$, figure 3.1 the vertices connected to i will be either within Ω^c or on the surface, $\partial\Omega^c$. This case is similar to figure 3.3 if we suppose there are no tetrahedra in Ω^c and we substitute $\Omega \to \Omega^c$, $\partial\Omega \to \partial\Omega^c$ in the figure. Here we will only have surface integrals on the internal side, to compute them we need to specify the derivatives, $\nabla\phi \cdot \hat{n}$. Since we require our solution to decay as $|\mathbf{x}| \to \infty$, if we choose $\partial\Omega^c$ very far from Ω we could set the values of the derivative to zero, but instead we can truncate this vacuum region closer to Ω and introduce the expected behaviour of the potentials derivative, for $d = 3$ we have [21, p. 184]

$$(\nabla\phi \cdot \hat{n})(\mathbf{x}) \approx \frac{1}{|\mathbf{x}|}\phi(\mathbf{x}) \;\text{ for }\; \mathbf{x} \in \partial\Omega^c \tag{3.53}$$

Assuming $|\mathbf{x}|$ is constant along the triangular element of tetrahedron e that also belongs to $\partial\Omega^c$ we suppose its the mid-point of triangle and call it $\mathbf{x}^e_{mid}$

$$\begin{aligned} s^{\partial}_{\phi,i} &= \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \int_{\partial\Omega^e} \frac{1}{|\mathbf{x}|}\phi_h(\mathbf{x}) N^e_\alpha dS \\ &= \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \frac{1}{|\mathbf{x}^e_{mid}|} \sum_{\beta=0}^{3} \phi^e_\beta \int_{\partial\Omega^e} N^e_\alpha N^e_\beta dS \\ &= \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \frac{1}{|\mathbf{x}^e_{mid}|} \sum_{\beta=0}^{2} \phi^s_\beta \times \begin{cases} \frac{1}{12}\Delta^e & \text{if } \alpha \neq \beta \\ \frac{1}{6}\Delta^e & \text{if } \alpha = \beta \end{cases} \quad \text{if } i \in \partial\Omega^c \end{aligned} \tag{3.54}$$

$$\text{with IMR:} \quad s^{\partial}_{\phi,i} = \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \frac{1}{|\mathbf{x}^e_{mid}|} \sum_{\beta=0}^{2} (\phi^s_\beta + \phi^s_{\beta,0}) \times \begin{cases} \frac{1}{24}\Delta^e & \text{if } \alpha \neq \beta \\ \frac{1}{12}\Delta^e & \text{if } \alpha = \beta \end{cases} \tag{3.55}$$

where we substituted the discretized potential ϕ_h, which collapse into an expansion on just three basis functions instead of four since we are computing ϕ_h on the surface of a tetrahedra. The areas Δ^e are for the triangular elements belonging to $\partial\Omega^c$ and the element e. From the n_q terms $s^{\partial}_{\phi,i}$ we build the final part of (3.35).

3.4.1 Jacobian of Poisson residual

Now that we have expressions for each part of Poisson's residual the derivatives can now be computed. Taking into account the signs in (3.36) and the expression that only depend on $\mathbf{m}$, (3.38) with IMR and (3.52), we have the Jacobian

$$\text{with IMR:} \quad Q^e_{\alpha\gamma} = \frac{\partial s^e_\alpha}{\partial \mathbf{m}^e_\gamma} = \frac{\Omega^e}{8}\nabla N^e_\gamma - \hat{n}^s \times \begin{cases} \frac{1}{24}\Delta^s & \text{if } \alpha \neq \gamma \\ \frac{1}{12}\Delta^s & \text{if } \alpha = \gamma \end{cases} \tag{3.56}$$

as we observer in (3.55), the second term only appears if (e,α) and (e,γ) both belong to $\partial\Omega$ This first Jacobian is a 3×1 vector, but as we shall see in Chap 5 it is more suitable having it as a row vector, hence we transpose (3.56), it is associated with the indices $i = n(e,\alpha)$ and $j = n(e,\gamma)$ that identify its location within a larger Jacobian matrix $5n_q \times 5n_q$ to be introduced later. This term only depends on j hence its the same for all i in the matrix column j.

The second Poisson's Jacobian is obtained applying $\frac{\partial}{\partial\phi}$ to $s^e_{\phi,\alpha}$, one of the parts (3.36), noticing s^e_α is linear function with respect to ϕ^e_γ, its coefficients are the derivatives, therefore from the stiffness term (3.43) and if BC (3.53) are used in (3.55) we have

$$-\frac{1}{2}K^e_{\alpha\gamma} + O^e_{\alpha\gamma} = \frac{\partial s^e_\alpha}{\partial \phi^e_\gamma} = \quad -\frac{1}{2}\nabla N^e_\alpha \cdot \nabla N^e_\gamma \Omega^e + \frac{1}{2}\frac{1}{|\mathbf{x}^e_{mid}|} \times \begin{cases} \frac{1}{24}\Delta^e & \text{if } \alpha \neq \gamma \text{ and } n(\alpha,\gamma) \in \partial\Omega^c \\ \frac{1}{12}\Delta^e & \text{if } \alpha = \gamma \text{ and } n(\alpha,\gamma) \in \partial\Omega^c \\ 0 & \text{if } n(\alpha,\gamma) \notin \partial\Omega^c \end{cases} \tag{3.57}$$

where the derivative turned $\phi^s_{\beta,0}$ and $\phi^e_{\beta,0}$ into zero, so there is no contribution. This term is a contribution to the entry of K matrix obtained previously, but now it includes as a second term the asymptotic boundary conditions.

3.5 Weak form for LLG equation

In an analogous way to what have been done with Poisson's residual, the weak form of LLG residual is obtained following the same procedure. We multiply LLG equation (3.1) with a basis function, integrate and rearranging we define

$$\mathbf{r}^e_\alpha(\mathbf{m},\phi) = \int_{\Omega^e} \left[\frac{\Delta \mathbf{m}_h}{\Delta t} + \mathbf{m}_h \times \left(\mathbf{h}_{ap} - \nabla\phi_h + \alpha_c \frac{\Delta \mathbf{m}_h}{\Delta t} - \nabla^2 \mathbf{m}_h\right)\right] N^e_\alpha(\mathbf{x}) d^3x \tag{3.58}$$

It is more complicated than the Poisson's residual, but we will separated it into three parts, the time derivative, the applied field together with the stray field and damping, and finally the exchange term. Some of the integrals involved will be approximated using the quadrature

from [10]. For a generic function $f(\mathbf{x})$, its integral in an element of volume Ω^e is approximated

$$\int_{\Omega^e} f(\mathbf{x}) d^3x \approx \sum_{\alpha=0}^{3} \frac{\Omega^e}{4} f(\mathbf{x}^e_\alpha) \tag{3.59}$$

Once we have $\mathbf{r}^e_\alpha$ and its Jacobians we can assemble the residuals at i as the following sum

$$\mathbf{r}_i = \sum_{\{(e,\alpha)|i=n(e,\alpha)\}} \mathbf{r}^e_\alpha \tag{3.60}$$

and an entry of the 3×3 or 3×1 Jacobians as

$$\frac{\partial \mathbf{r}_i}{\partial \mathbf{m}_j} = \sum_{\{(e,\alpha,\beta)|i=n(e,\alpha) \wedge j=n(e,\beta)\}} \frac{\partial \mathbf{r}^e_\alpha}{\partial \mathbf{m}^e_\beta} \tag{3.61}$$

$$\frac{\partial \mathbf{r}_i}{\partial \phi_j} = \sum_{\{(e,\alpha,\beta)|i=n(e,\alpha) \wedge j=n(e,\beta)\}} \frac{\partial \mathbf{r}^e_\alpha}{\partial \phi^e_\beta} \tag{3.62}$$

The computation of such sums will be one of the subjects for Chapter 5, for now we will just derive their factors from the expression (3.58).

1. The time derivative

The time derivative residual of (3.58) after discretization by finite differences is given by

$$\begin{aligned} \mathbf{r}^{e,A}_\alpha &= \int_{\Omega^e} \frac{\mathbf{m}_h - \mathbf{m}_{h,0}}{\Delta t} N^e_\alpha d^3x \\ &= \frac{\Omega^e}{4} \sum_{\beta=0}^{3} \frac{\mathbf{m}_h(\mathbf{x}^e_\beta) - \mathbf{m}_{h,0}(\mathbf{x}^e_\beta)}{\Delta t} N^e_\alpha(\mathbf{x}^e_\alpha) \\ &= \frac{\Omega^e}{4} \frac{\mathbf{m}_h(\mathbf{x}^e_\alpha) - \mathbf{m}_{h,0}(\mathbf{x}^e_\alpha)}{\Delta t} \end{aligned} \tag{3.63}$$

where the integral was approximated using (3.59). The Jacobian is simply given by

$$A^e_{\alpha,\alpha} = \frac{\partial \mathbf{r}^{e,A}_\alpha}{\partial \mathbf{m}^e_\alpha} = I_{3\times 3} \frac{1}{\Delta t} \frac{\Omega^e}{4} \tag{3.64}$$

using the fact $\frac{\partial \mathbf{m}^e_\alpha}{\partial \mathbf{m}^e_\alpha} = I_{3\times 3}$ and (3.29). This term fills the diagonals of the 3×3 blocks associated with (i, i) with a multiple of the constant term $\frac{1}{\Delta t}\frac{\Omega^e}{4}$. That multiple is the number of connections of a given vertex with its neighbours.

2. Applied field, Stray field and Damping Jacobian, $D^e_{\alpha,\gamma}$

The residual from LLG is

$$\mathbf{r}^{e,D}_\alpha = \int_{\Omega^e} \left(\frac{\mathbf{m}^e_h + \mathbf{m}^e_{h,0}}{2} \right) \times \left(\frac{\mathbf{h}_{ap} + \mathbf{h}_{ap,0}}{2} - \nabla \left(\frac{\phi^e_h + \phi^e_{h,0}}{2} \right) - \alpha_c \frac{\mathbf{m}^e_h - \mathbf{m}^e_{h,0}}{\Delta t} \right) N^e_\alpha d^3x \tag{3.65}$$

Using the integral approximation (3.59) we get

$$\mathbf{r}_{\alpha}^{e,D} = \left(\frac{\mathbf{m}_{\alpha}^{e} + \mathbf{m}_{\alpha,0}^{e}}{2}\right) \times \left(\frac{\mathbf{h}_{ap,\alpha}^{e} + \mathbf{h}_{ap,\alpha,0}^{e}}{2} - \frac{1}{2}\sum_{\beta=0}^{3}(\phi_{\beta}^{e} + \phi_{\beta,0}^{e})\nabla N_{\beta}^{e} - \alpha_c \frac{\mathbf{m}_{\alpha}^{e} - \mathbf{m}_{\alpha,0}^{e}}{\Delta t}\right)\frac{\Omega^e}{4} \tag{3.66}$$

Observe that the magnetization is evaluated at $\mathbf{x}_{\alpha}^{e}$ while the gradient of the potential remains unaffected by the integration since it does not depend on $\mathbf{x}$. This residual term has a dependence on magnetization $\mathbf{m}_h$ and potential field ϕ_h, two derivatives have to be computed. The first one is given by

$$D_{\alpha,\gamma}^{e} = \frac{\partial \mathbf{r}_{\alpha}^{e}}{\partial \mathbf{m}_{\gamma}^{e}} = \frac{\partial}{\partial \mathbf{m}_{\gamma}^{e}} \int_{\Omega^e} \left(\frac{\mathbf{m}_{h}^{e} + \mathbf{m}_{h,0}^{e}}{2}\right) \times \left(\frac{\mathbf{h}_{ap} + \mathbf{h}_{ap,0}}{2} - \nabla\left(\frac{\phi_{h}^{e} + \phi_{h,0}^{e}}{2}\right) - \alpha \frac{\mathbf{m}_{h}^{e} - \mathbf{m}_{h,0}^{e}}{\Delta t}\right) N_{\alpha}^{e} d^3x \tag{3.67}$$

and we will compute $\frac{\partial \mathbf{r}_{\alpha}^{e}}{\partial \mathbf{m}_{\gamma}^{e}}$ by components.

Using Einstein summation for lower indices, and knowing $[\mathbf{a} \times \mathbf{b}]_i = \epsilon_{ijk} a_j b_k$, we observe that the rate of change of the $i \in \{x, y, z\}$ component of $\mathbf{r}_{\alpha}^{e}$ at vertex $n(e, \alpha)$ with respect to the $l \in \{x, y, z\}$ component of magnetization at vertex $n(e, \gamma)$ is given by

$$\frac{\partial r_{\alpha,i}}{\partial m_{\gamma,l}} = \frac{\partial}{\partial m_{\gamma,l}} \int_{\Omega^e} \epsilon_{ijk} \frac{m_{h,j} + m_{h,j,0}}{2} \left(\frac{h_{ap,k} + h_{ap,k,0}}{2} - \frac{\partial}{\partial x_k}\frac{\phi_h + \phi_{h,0}}{2} - \alpha \frac{m_{h,k} - m_{h,k,0}}{\Delta t}\right) N_{\alpha} d^3x \tag{3.68}$$

where we dropped the index e. From expansions for $\mathbf{m}_h$ and ϕ_h, the derivatives are

$$\begin{aligned} \frac{\partial m_{h,k}}{\partial m_{\gamma,l}} &= \delta_{kl} N_{\gamma}^{e} \\ \frac{\partial \phi_h}{\partial x_k} &= \phi_{\beta}^{e} \frac{\partial N_{\beta}^{e}}{\partial x_k} \end{aligned} \tag{3.69}$$

using the product rule on (3.68) and substituting (3.69)

$$\frac{\partial r_{\alpha,i}}{\partial m_{\gamma,l}} = \int_{\Omega^e} \epsilon_{ink} \frac{N_{\alpha}^{e} N_{\gamma}^{e}}{2} \left(\frac{h_{ap,k} + h_{ap,k,0}}{2} - \frac{\phi_{\beta} + \phi_{\beta,0}}{2}\frac{\partial N_{\beta}^{e}}{\partial x_k} - \alpha \frac{m_{h,k} - m_{h,k,0}}{\Delta t}\right) \tag{3.70}$$

$$- \alpha \epsilon_{ijl} \frac{N_{\alpha}^{e} N_{\gamma}^{e}}{\Delta t} \frac{m_{h,j} + m_{h,j,0}}{2} d^3x \tag{3.71}$$

Now switching indices $\epsilon_{ijn} = -\epsilon_{inj}$ and renaming the index j as k we can factor out ϵ_{ilk} in both terms, combining them we have

$$\frac{\partial r_{\alpha,i}}{\partial m_{\gamma,n}} = -\int_{\Omega^e} \epsilon_{ink} N_{\alpha}^{e} N_{\gamma}^{e} \left(-\frac{h_{ap,k} + h_{ap,k,0}}{4} + \frac{\phi_{\beta} + \phi_{\beta,0}}{4}\frac{\partial N_{\beta}^{e}}{\partial x_k} - \alpha \frac{m_{h,k,0}}{\Delta t}\right) d^3x \tag{3.72}$$

In our case the integrand in (3.72) will be evaluated at each $\mathbf{x}_{\alpha}^{e}$ of tetrahedron e giving

$$\frac{\partial r_{\alpha,i}^{e}}{\partial m_{\gamma,l}^{e}} \approx -\epsilon_{ilk} \sum_{\beta=0}^{3} N_{\alpha}^{e}(\mathbf{x}_{\beta}^{e}) N_{\gamma}^{e}(\mathbf{x}_{\beta}^{e}) v_k(\mathbf{x}_{\beta}^{e}) \frac{\Omega^e}{4} \tag{3.73}$$

where v_k with $k \in \{x, y, z\}$ corresponds to the three terms in parenthesis in (3.72). The

summation terms are only nonzero when $\alpha = \beta = \gamma$ therefore we have

$$\frac{\partial r^e_{\alpha,i}}{\partial m^e_{\alpha,l}} = -\epsilon_{ilk}\left(-\frac{h_{ap,k} + h_{ap,k,0}}{4} + \frac{\phi_\beta + \phi_{\beta,0}}{4}\frac{\partial N^e_\beta}{\partial x_k} - \alpha_c \frac{m_{h,k,0}}{\Delta t}\right)_{(\mathbf{x}^e_\alpha)} \frac{\Omega^e}{4} \tag{3.74}$$

This calculation can be represented in vectorial form by observing first what the operation $-\epsilon_{ilk}$ does to the factor in parenthesis in (3.74), that we define as v_k

Construction the following 3×3 matrix block

$$\begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix} \tag{3.75}$$

The specification of an entry in this skew matrix block is made with $i, l \in \{x, y, z\}$ in (3.74). The tensor operation $-\epsilon_{ilk}$ transforms v_k into an entry of this matrix block 3×3, and the block in turn is associated with $(n(e,\alpha), n(e,\alpha))$. As we shall see, these blocks are diagonal blocks in the larger Jacobian matrix, meaning the residual at vertex $n(e,\alpha)$ only varies when the magnetization at this vertex varies as well and does not depend on neighbour vertices.

In vectorial notation, the (3.75) block is given by

$$D^e_{\alpha,\alpha} = \frac{\partial \mathbf{r}^e_\alpha}{\partial \mathbf{m}^e_\alpha} = \frac{\Omega^e}{4}\mathbf{skew}\left[-\frac{\mathbf{h}_{ap}(\mathbf{x}^e_\alpha) + \mathbf{h}_{ap,0}(\mathbf{x}^e_\alpha)}{4} + \sum_{\beta=0}^{3}\frac{\phi^e_\beta + \phi^e_{\beta,0}}{4}\nabla N^e_\beta - \frac{\alpha_c}{\Delta t}\mathbf{m}_{h,0}(\mathbf{x}^e_\alpha)\right] \tag{3.76}$$

where the **skew** operator maps the three entries of $\mathbf{v}$ into the 3×3 skew-symmetric matrix given in (3.75).

Using similar techniques to the previous derivative we have

$$\begin{aligned} P^e_{\alpha,\beta} = \frac{\partial \mathbf{r}^e_\alpha}{\partial \phi^e_\gamma} &= -\frac{\partial}{\partial \phi^e_\gamma}\int_{\Omega^e}\left(\frac{\mathbf{m}^e_h + \mathbf{m}^e_{h,0}}{2}\right) \times \nabla\left(\frac{\phi^e_h + \phi^e_{h,0}}{2}\right) N^e_\alpha(\mathbf{x}^\mathbf{e}) d^3x = \\ &= -\frac{1}{4}\int_{\Omega^e} N^e_\alpha(\mathbf{x}^e)(\mathbf{m}^e_h + \mathbf{m}^e_{h,0}) \times \sum_{\beta=0}^{3}\frac{\partial \phi^e_\beta}{\partial \phi^e_\gamma}\nabla N^e_\beta d^3x = \\ &= -\frac{1}{4}\int_{\Omega^e} N^e_\alpha(\mathbf{x}^e)(\mathbf{m}^e_h + \mathbf{m}^e_{h,0}) \times \nabla N^e_\gamma d^3x \\ &\approx -\frac{1}{4}\sum_{\eta=0}^{3}\frac{\Omega^e}{4} N^e_\alpha(\mathbf{x}^e_\eta)(\mathbf{m}^e_h(\mathbf{x}^e_\eta) + \mathbf{m}^e_{h,0}(\mathbf{x}^e_\eta)) \times \nabla N^e_\gamma \\ &= -\frac{\Omega^e}{16}(\mathbf{m}^e_\alpha + \mathbf{m}^e_{\alpha,0}) \times \nabla N^e_\gamma \end{aligned} \tag{3.77}$$

$P^e_{\alpha,\beta}$ will contribute for a 3×1 column vector dependent on both $i = n(e,\alpha)$ and $j = n(e,\gamma)$.

3. Exchange Jacobian, $E^e_{\alpha,\gamma}$

To compute the Jacobian of the exchange term in equation (3.58) we will treat one of the residual components, i. Using summation convention on i, j, k and p, q, all belonging to

$\{x, y, z\}$ and dropping the upper index e, we start integrating by parts and substitute $\mathbf{m}_h$

$$r^e_{\alpha,i} = \int_{\Omega^e} \left[(\mathbf{m} \times \nabla^2 \mathbf{m})\right]_i N_\alpha d^3x = -\int_{\Omega^e} \epsilon_{ijk} \frac{\partial (m_{h,j} N_\alpha)}{\partial x_p} \frac{\partial m_{h,k}}{\partial x_p} d^3x + \int_{\partial\Omega^e} \epsilon_{ijk} N^e_\alpha m_{h,j} \frac{\partial m_{h,k}}{\partial x_q} n_q dS \tag{3.78}$$

If we assume Brown's second equilibrium condition (2.45), the surface integral integrand is zero

$$\epsilon_{ijk} m_{h,j} \frac{\partial m_{h,k}}{\partial x_q} n_q = \epsilon_{ijk} m_{h,j} \frac{\partial m_{h,k}}{\partial \hat{n}} = \left[\mathbf{m}_h \times \frac{\partial \mathbf{m}_h}{\partial \hat{n}}\right]_i = 0 \tag{3.79}$$

Using the derivative of a product rule on the first integral of (3.78) we have

$$r_{\alpha,i} = -\int_{\Omega^e} \epsilon_{ijk} m_{h,j} \frac{\partial N_\alpha}{\partial x_p} \frac{\partial m_{h,k}}{\partial x_p} d^3x - \int_{\Omega^e} \epsilon_{ijk} N_\alpha \frac{\partial m_{h,j}}{\partial x_p} \frac{\partial m_{h,k}}{\partial x_p} d^3x \tag{3.80}$$

The second integral is of the form $\mathbf{a} \times \mathbf{a} = 0$. One of the components $k \in \{x, y, z\}$ of the expansion $\mathbf{m}_h$ is given by

$$m_{h,k} = \sum_{\beta=0}^{3} m_{\beta,k} N_\beta = m_{\beta,k} N_\beta \tag{3.81}$$

After substituting (3.81) into the first integral in (3.80)

$$r_{\alpha,i} = -\int_{\Omega^e} \epsilon_{ijk} m_{\beta,j} m_{\eta,k} \frac{\partial N_\alpha}{\partial x_p} \frac{\partial N_\eta}{\partial x_p} N_\beta d^3x \tag{3.82}$$

Except for N_β no other factor depend on $\mathbf{x}$ hence using either (3.29) or the integral using (3.59) we have after recovering the sums

$$r_{\alpha,i} = -\sum_{\eta=0}^{3} \left(\sum_{\beta=0}^{3} \mathbf{m}_\beta \times \mathbf{m}_\eta\right)_i \nabla N_\alpha \cdot \nabla N_\eta \frac{\Omega^e}{4} = -\sum_{\eta=0}^{3} (\mathbf{m}_{tot} \times \mathbf{m}_\eta)_i \nabla N_\alpha \cdot \nabla N_\eta \frac{\Omega^e}{4} \tag{3.83}$$

where we make the important observation that entries from the Poisson's stiffness matrix (3.44) are also used in the construction of the exchange residual and $\mathbf{m}_{tot}$ for a given element is

$$\mathbf{m}^e_{tot} = \sum_{\beta=0}^{3} \mathbf{m}^e_\beta \tag{3.84}$$

Hence renaming the dummy variable η as β and joining the three i components into a vector

$$\mathbf{r}^e_\alpha = -\frac{1}{4} \mathbf{m}^e_{tot} \times \left(\sum_{\beta=0}^{3} \mathbf{m}^e_\beta K^e_{\alpha,\beta}\right) \tag{3.85}$$

$$\text{with IMR: } \mathbf{r}^e_\alpha = -\frac{1}{4} \frac{\mathbf{m}^e_{tot} + \mathbf{m}^e_{tot,0}}{2} \times \left(\sum_{\beta=0}^{3} \frac{\mathbf{m}^e_\beta + \mathbf{m}^e_{\beta,0}}{2} K^e_{\alpha,\beta}\right) \tag{3.86}$$

where we included the Ω^e in $K^e_{\alpha,\beta}$.

To compute the Jacobian we take derivative $\frac{\partial}{\partial m^e_{\gamma,n}}$ of (3.82), using the product rule

$$\frac{\partial r_{\alpha,i}}{\partial m_{\gamma,n}} = -\epsilon_{ijk}\int_{\Omega^e} N_\beta \frac{\partial N_\alpha}{\partial x_p}\frac{\partial N_\eta}{\partial x_p}\left[\frac{\partial m_{\beta,j}}{\partial m_{\gamma,n}} m_{\eta,k} + m_{\beta,j}\frac{\partial m_{\eta,k}}{\partial m_{\gamma,n}}\right] d^3x \tag{3.87}$$

Substituting $\frac{\partial m_{\beta,j}}{\partial m_{\gamma,n}} = \delta_{\beta\gamma}\delta_{jn}$ and $\frac{\partial m_{\eta,k}}{\partial m_{\gamma,n}} = \delta_{\eta\gamma}\delta_{kn}$, replacing $\epsilon_{ijn} = -\epsilon_{inj}$ and renaming j as k

$$\frac{\partial r_{\alpha,i}}{\partial m_{\gamma,n}} = -\epsilon_{ink}\int_{\Omega^e} N_\beta \frac{\partial N_\alpha}{\partial x_p}\frac{\partial N_\eta}{\partial x_p}\left[m_{\eta,k}\delta_{\beta\gamma} - m_{\beta,k}\delta_{\eta\gamma}\right] d^3x \tag{3.88}$$

Changing the basis indices applying the Kronecker delta's we have after rearranging

$$\frac{\partial r_{\alpha,i}}{\partial m_{\gamma,n}} = -\epsilon_{ink}\int_{\Omega^e} \frac{\partial N_\alpha}{\partial x_p}\left[m_{\eta,k}N_\gamma\frac{\partial N_\eta}{\partial x_p} - m_{\beta,k}N_\beta\frac{\partial N_\gamma}{\partial x_p}\right] d^3x \tag{3.89}$$

$$= -\epsilon_{ink}\int_{\Omega^e} \left[m_{\eta,k}N_\gamma\frac{\partial N_\alpha}{\partial x_p}\frac{\partial N_\eta}{\partial x_p} - m_{\beta,k}N_\beta\frac{\partial N_\alpha}{\partial x_p}\frac{\partial N_\gamma}{\partial x_p}\right] d^3x \tag{3.90}$$

On the both integrals only N^e_γ and N^e_β are dependent on $\mathbf{x}$, its integration gives $\frac{\Omega^e}{4}$ factor. We conclude that the component $i \in \{x, y, z\}$ of the 3×3 Jacobian block associated with the indices $(n(e,\alpha), n(e,\gamma))$ is given by

$$\frac{\partial r_{\alpha,i}}{\partial m_{\gamma,n}} = -\epsilon_{ink}\left[m_{\eta,k}\frac{\partial N_\alpha}{\partial x_p}\frac{\partial N_\eta}{\partial x_p} - m_{tot,k}\frac{\partial N_\alpha}{\partial x_p}\frac{\partial N_\gamma}{\partial x_p}\right]\frac{\Omega^e}{4} \tag{3.91}$$

We can rewrite this equation using the stiffness matrix as

$$\frac{\partial r^e_{\alpha,i}}{\partial m^e_{\gamma,n}} = -\frac{\epsilon_{ink}}{4}\left[\sum_{\beta=0}^{3} m^e_{\beta,k}K^e_{\alpha,\beta} - m^e_{tot,k}K^e_{\alpha,\gamma}\right] \tag{3.92}$$

Each component of the vector in parenthesis is mapped into the skew matrix by $-\epsilon_{ink}$, we write then the 3×3 block associated with the indices $i = n(e,\alpha)$ and $j = n(e,\beta)$ as

$$E^e_{\alpha,\gamma} = \mathbf{skew}\left[\sum_{\beta=0}^{3}\mathbf{m}^e_\beta\frac{K^e_{\alpha,\beta}}{4} - \mathbf{m}^e_{tot}\frac{K^e_{\alpha,\gamma}}{4}\right] \tag{3.93}$$

$$\text{with IMR: } E^e_{\alpha,\gamma} = \mathbf{skew}\left[\sum_{\beta=0}^{3}\frac{\mathbf{m}^e_\beta + \mathbf{m}^e_{\beta,0}}{2}\frac{K^e_{\alpha,\beta}}{4} - \frac{\mathbf{m}^e_{tot} + \mathbf{m}^e_{tot,0}}{2}\frac{K^e_{\alpha,\gamma}}{4}\right] \tag{3.94}$$

where the Poisson's stiffness entries are again used in this term.

3.6 Conclusion

In this Chaper we laid out the plan to solve LLG and Poisson's equation. We first discretized time with increment Δt, using Implicit Mid-Point Rule: derivatives with respect to time were replaced by finite differences and function dependencies on time, magnetization or potential were replaced by the average between two successive instants of time. Then space was

discretized. Permeating space with a $d = 3$ mesh with n_q vertices that defined n_e tetrahedral elements, allowed us to introduce four basis functions, N^e_α, for each one, and for surface triangular elements s, the basis functions N^s_α. The potential and magnetization were then approximated by expanding them in this nonorthogonal basis as in (3.27) and (3.28), which upon substitution the strong equations LLG and Poisson, showed us these cannot be solutions.

In response, we multiplied them by a basis function and integrated by parts on Ω^e, we then obtained their form without second derivatives and only gradients, allowing now linear solutions.

In order to solve these weak equations for each vertex i we introduced Newton-Raphson method which for a given initial configuration linear approximates the residuals of both equations for each i yielding a linear system of equations where each equation is associated with a vertex of the mesh. Iterative search for zero residuals with this linear system progressively gives us better solutions for the configuration for $t + \Delta t$ that satisfy both weak forms.

This system of equations is formed by the Jacobians of residuals for both equations, we derived explicit expressions for Poisson's residual and for LLG residual, the time derivative Jacobian, the applied field, stray field and damping, and finally the exchange Jacobians. These depend on whether vertex i is in Ω or Ω^c or the boundaries. The construction of these Jacobians depend on having first a mesh for $\Omega \cup \Omega^c$. We observe that Poisson's equation was the only term requiring a mesh in Ω^c since we have asymptotic behavior to be satisfied.

What if we could avoid meshing Ω^c? Then we would solve the LLG equation and Poisson's equation focusing only on the system. In the next chapter this is what we will do introducing how equations from the Boundary Element Method help to solve the FEM problem for Poisson's equation.

Chapter 4

Finite Element Method and Boundary Element Method for Poisson's Equation

4.1 Introduction

This chapter is about the Poisson's problem. The goal is to reformulate it in such a way as to replace the asymptotic boundary conditions by Dirichlet boundary conditions on the boundary of the system, $\partial\Omega$. As a consequence we will not need the system surrounding, Ω^c, in computing the evolution of the magnetization and potential.

The central idea is to introduce a new potential, u, that can be computed with FEM only on Ω, and whose boundary values can be mapped into the ϕ at the boundary as well, a strategy used in the Boundary Element Method, [29, 30]. By knowing ϕ on $\partial\Omega$, we only have to solve Poisson's equation in Ω.

4.2 A new potential, u

We start by splitting ϕ as $\phi = u + v$ and substitute in both (3.4) and the matching conditions (3.6) we get two new equations. Defining conditions on the potential u [30], conditions on v are obtained. Suppose that

$$\begin{cases} \nabla^2 u = \nabla \cdot \mathbf{m} & \text{if } \mathbf{x} \in \Omega \\ u = 0 & \text{if } \mathbf{x} \in \Omega^c \end{cases} \tag{4.1}$$

with Neumann boundary condition

$$\frac{\partial u_{int}}{\partial n} = \mathbf{m} \cdot \hat{n} \text{ if } \mathbf{x} \in \partial\Omega \tag{4.2}$$

As a consequence of (3.6), the asymptotic behaviour for ϕ and the assumption on u above, a few lines of algebra will show that the potential v has to satisfy

$$\nabla^2 v = 0 \text{ for } \mathbf{x} \in \Omega \cup \Omega^c \tag{4.3}$$

$$\frac{\partial v_{int}}{\partial n} - \frac{\partial v_{ext}}{\partial n} = 0 \text{ for } \mathbf{x} \in \partial\Omega \tag{4.4}$$

$$v_{int} - v_{ext} = -u_{int} \text{ for } \mathbf{x} \in \partial\Omega \tag{4.5}$$

$$v(\mathbf{x}) \to 0 \text{ as } |\mathbf{x}| \to \infty \tag{4.6}$$

The goal is to establish a relation between $\phi_{int}(\mathbf{x})$ and $u_{int}(\mathbf{x})$ on the boundary $\partial\Omega$. The strategy will be to build an integral equation to obtain $v_{int}(\mathbf{x})$ from $u_{int}(\mathbf{x})$ using (4.5) and then substitute into $\phi_{int} = u_{int} + v_{int}$.

The first step is to relate all values of $v(\mathbf{x})$ for $\mathbf{x} \in \Omega$ with all $v^{int}(\mathbf{x})$ using an integral equation and for that we will use the "free-space Green function". This Green function will satisfy the fundamental equation [3, p. 251]

$$\nabla'^2 G(\mathbf{x}, \mathbf{x}') = -\delta(\mathbf{x} - \mathbf{x}') \text{ for } \mathbf{x}, \mathbf{x}' \in \mathbb{R}^3 \tag{4.7}$$

and asymptotic conditions $G(\mathbf{x}, \mathbf{x}') \to 0$ as $|\mathbf{x} - \mathbf{x}'| \to \infty$. The solution of this PDE can be obtained using the divergence theorem, for $d = 3$ we have [3, p. 252]

$$G(\mathbf{x}, \mathbf{x}') = \frac{1}{4\pi} \frac{1}{|\mathbf{x} - \mathbf{x}'|} \text{ for } \mathbf{x}, \mathbf{x}' \in \mathbb{R}^3 \tag{4.8}$$

We now define the integral equation for (4.3) similarly as done in FEM, (3.32). While in FEM, the weight functions were the shape functions, $N^e_\alpha(\mathbf{x})$, in BEM, the weight function is the free-space Green function above. From (4.3) we multiply by (4.8) and integrate

$$\int_\Omega \nabla'^2 v(\mathbf{x}') G(\mathbf{x}, \mathbf{x}') d^3x' = 0 \tag{4.9}$$

In an analogous way to FEM we now use integration by parts (3.32) giving us

$$-\int_\Omega \nabla' v(\mathbf{x}') \cdot \nabla' G(\mathbf{x}, \mathbf{x}') d^3x' + \int_{\partial\Omega} G(\mathbf{x}, \mathbf{x}') \nabla' v(\mathbf{x}') \cdot \hat{n}' dS' = 0 \tag{4.10}$$

But now in BEM, we proceed by using integration by parts again to introduce a new surface integral

$$\int_\Omega v(\mathbf{x}') \nabla'^2 G(\mathbf{x}, \mathbf{x}') d^3x' - \int_{\partial\Omega} v(\mathbf{x}') \nabla' G(\mathbf{x}, \mathbf{x}') \cdot \hat{n}' dS' + \int_{\partial\Omega} G(\mathbf{x}, \mathbf{x}') \nabla' v(\mathbf{x}') \cdot \hat{n}' dS' = 0 \tag{4.11}$$

It is the first integral in (4.11) that will require more attention. If $\mathbf{x} \in \Omega \setminus \partial\Omega$ then using (4.7) and the Dirac Delta function property

$$\int_\Omega f(\mathbf{x}') \delta(\mathbf{x} - \mathbf{x}') d^3x' = f(\mathbf{x}) \text{ if } \mathbf{x} \in \Omega \setminus \partial\Omega \tag{4.12}$$

the first integral yields $v(\mathbf{x})$ and the expression

$$v(\mathbf{x}) = -\int_{\partial\Omega} v(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' + \int_{\partial\Omega} G(\mathbf{x},\mathbf{x}')\nabla' v(\mathbf{x}')\cdot\hat{n}'dS' \text{ if } \mathbf{x}\in\Omega\setminus\partial\Omega \text{ or } \Omega^c \quad (4.13)$$

This expression uses information of $v(\mathbf{x})$ and $\nabla v(\mathbf{x})$ on the boundary for (int) or (ext) values to compute $v(\mathbf{x})$ within the system Ω or outside. It is possible from surface information to evaluate interior or exterior points because these two surface integrals are in fact a reformulation of the volume integral in (4.11), and this integral is of the form (4.12). However what we want is to relate the boundary values $v_{int}(\mathbf{x})$ to substitute into MC, (4.5). Since on the boundary the Delta function property cannot be used, this suggests that we extend the Ω region to include $\mathbf{x}$, let it be a semi-sphere centered at $\mathbf{x}$ and radius ϵ such as illustrated in figure 4.1-I. Following [29, p. 49-51], we first relate the $v_{int}(\mathbf{x})$ values by evaluating (4.11) at

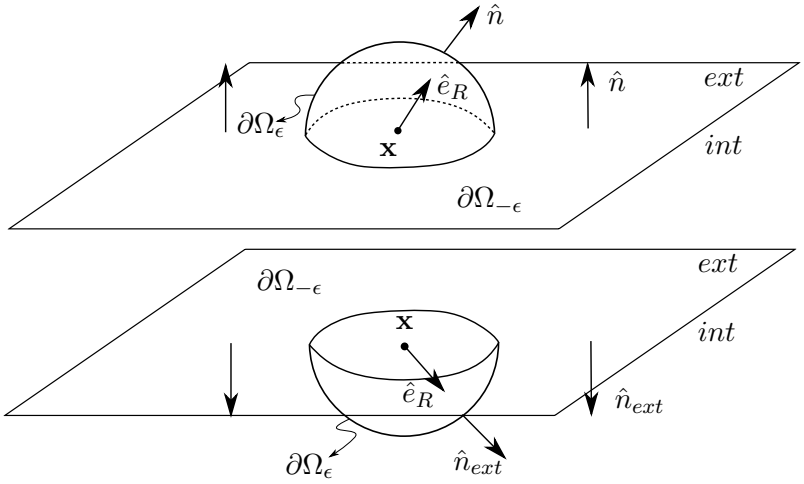

Figure 4.1: Locally at $\mathbf{x}$ of figure 3.1 from previous Chapter. Top figure (I) represents the extension of Ω to include the boundary point $\mathbf{x}$; The lower picture (II) represents the extension of Ω^c to include it. In both we show the case where the surface $\partial\Omega$ is smooth, so locally the surface looks like a plane and the semi-spheres with radius ϵ are half-spheres, both determine two complementary solid angles of 2π.

$\mathbf{x}\in\partial\Omega$. Then we extend Ω to include $\mathbf{x}$ using a semi-sphere. Now (4.12) can be used on the first integral. We define the semi-sphere surface as $\partial\Omega_\epsilon$ and the remaining surface with a hole as $\partial\Omega_{-\epsilon}$, see figure 4.1-I. Using the Delta function property on the first integral in (4.11) we get

$$v_{int}(\mathbf{x}) = -\int\limits_{\partial\Omega_\epsilon\cup\partial\Omega_{-\epsilon}} v_{int}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' + \int\limits_{\partial\Omega_\epsilon\cup\partial\Omega_{-\epsilon}} G(\mathbf{x},\mathbf{x}')\nabla' v_{int}(\mathbf{x}')\cdot\hat{n}'dS' \quad (4.14)$$

Both integrals can be split into two integrals: on $\partial\Omega_\epsilon$ and $\partial\Omega_{-\epsilon}$. We will compute first the one on the semi-spherical region, $\partial\Omega_\epsilon$. Defining $R=|\mathbf{x}-\mathbf{x}'|$ and using spherical coordinates we have $\hat{e}_R$ parallel to the outward normal to the surface of Ω as in figure (4.1), and from $\nabla' = \nabla_{\mathbf{x}'-\mathbf{x}} = \hat{e}_R\frac{\partial}{\partial R} + \hat{e}_\theta\frac{1}{R}\frac{\partial}{\partial\theta} + \hat{e}_\phi\frac{1}{Rsin(\theta)}\frac{\partial}{\partial\phi}$ we have $\hat{n}\cdot\nabla_{\mathbf{x}'-\mathbf{x}} = \frac{\partial}{\partial R}$. The first integral is then

given by

$$\begin{aligned}
&\int_{\partial\Omega_\epsilon} v_{int}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' \\
&= \int_{\partial\Omega_\epsilon} v_{int}(\mathbf{x}')\frac{\partial}{\partial R}\frac{1}{4\pi}\frac{1}{R}dS' \\
&= -\frac{1}{4\pi R^2}\int_{\partial\Omega_\epsilon} v_{int}(\mathbf{x}')dS' \\
&\approx -\frac{A(\mathbf{x})}{4\pi R^2}v_{int}(\mathbf{x}) \qquad (4.15)
\end{aligned}$$

$A(\mathbf{x})$ is the surface area of the semi-sphere that extended Ω into Ω^c. Observe that $\frac{A}{R^2}$ is the solid angle, figure 4.1 shows an example for a smooth surface, the solid angle is then 2π. Since (4.15) does not depend on R then it is valid for any R, therefore

$$\lim_{\epsilon\to 0}\int_{\partial\Omega_\epsilon} v_{int}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' = -\gamma_\cap(\mathbf{x})v_{int}(\mathbf{x}) \qquad (4.16)$$

where $\gamma_\cap(\mathbf{x}) = \frac{A(\mathbf{x})}{4\pi R^2}$.

For the second integral along $\partial\Omega_\epsilon$ we have

$$\begin{aligned}
\int_{\partial\Omega_\epsilon} G(\mathbf{x},\mathbf{x}')\nabla' v_{int}(\mathbf{x}')\cdot\hat{n}'dS' &= \int_{\partial\Omega_\epsilon} -\frac{1}{4\pi}\frac{1}{|\mathbf{x}'-\mathbf{x}|}\nabla' v_{int}(\mathbf{x}')\cdot\hat{n}'dS' \\
&= -\frac{1}{4\pi}\frac{1}{R}\int_{\partial\Omega_\epsilon}\nabla' v_{int}(\mathbf{x}')\cdot\hat{n}'dS' \\
&\approx \frac{R}{2}\nabla v_{int}(\mathbf{x})\cdot\hat{n} \qquad (4.17)
\end{aligned}$$

Taking the limit we have

$$\lim_{\epsilon\to 0}\int_{\partial\Omega_\epsilon} G(\mathbf{x},\mathbf{x}')\nabla' v_{int}(\mathbf{x}')\cdot\hat{n}'dS' = 0 \qquad (4.18)$$

Since both the third and forth integrals are well behaved [29, p. 51] then we have the integration along $\partial\Omega_{-\epsilon} \to \partial\Omega$. We now reformulate (4.14) with (4.16) and (4.18) as

$$v_{int}(\mathbf{x})\,(1-\gamma_\cap(\mathbf{x})) = -\int_{\partial\Omega} v_{int}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' + \int_{\partial\Omega} G(\mathbf{x},\mathbf{x}')\nabla' v_{int}(\mathbf{x}')\cdot\hat{n}'dS' \quad (4.19)$$

By comparing the case where $\mathbf{x}$ is inside Ω in (4.13) with this expression, we can qualitatively interpret that when the Delta function is on the boundary "part of it" is outside Ω, so "only part" of $v_{int}(\mathbf{x})$ is picked when computing the two surface integrals in (4.19) while in (4.13) all of $v_{int}(\mathbf{x})$ is the output. How much the Delta function is left out of Ω is measured by the area of the semi-sphere outside the original Ω with respect to the total area of the unit sphere, this is what $\gamma_\cap(\mathbf{x})$ measures in (4.19).

Now we need to obtain an analogous relation for $v_{ext}(\mathbf{x})$, we will start with Ω^c still finite. Noticing that it is not a Jordan region [31, p. 996] the divergence theorem cannot be applied, so we divide it into two Jordan regions: Ω_1^c and Ω_2^c, as in figure 4.2, we can now apply the divergence theorem and also integration by parts. Now choosing the same point $\mathbf{x}$ on the

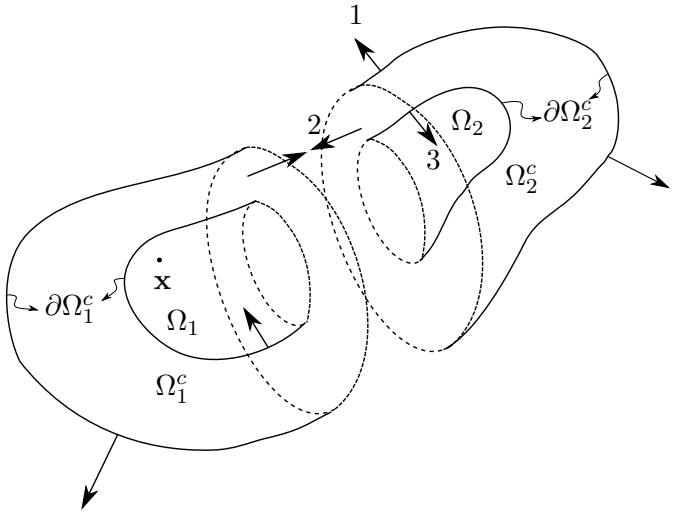

Figure 4.2: Two Jordan regions: Ω_1^c, Ω_2^c. The vectors are normal to the surface and the sequence 1,2,3 shows part of the outline of boundary $\partial\Omega_2^c$

surface now defined as $\partial\Omega_1^c$ we extend Ω_1^c to contain it as in figure 4.1-II, the first integral in (4.11) is well defined. We then obtain for Ω_1^c an analogous expression of (4.14) where the boundary is again split into the semi-sphere $\partial\Omega_{1,\epsilon}^c$ and the rest as $\partial\Omega_{1,-\epsilon}^c$.

$$v_{ext}(\mathbf{x}) = -\int\limits_{\partial\Omega_{1,\epsilon}^c \cup \partial\Omega_{1,-\epsilon}^c} v_{ext}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'_{ext}dS' + \int\limits_{\partial\Omega_{1,\epsilon}^c \cup \partial\Omega_{1,-\epsilon}^c} G(\mathbf{x},\mathbf{x}')\nabla' v_{ext}(\mathbf{x}')\cdot\hat{n}'_{ext}dS' \tag{4.20}$$

The integral relation for Ω_2^c is also of the form (4.11) but since $\mathbf{x}$ is on Ω_1^c the first integral is zero, rearranging we have

$$0 = -\int\limits_{\partial\Omega_{2,\epsilon}^c \cup \partial\Omega_{2,-\epsilon}^c} v_{ext}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'_{ext}dS' + \int\limits_{\partial\Omega_{2,\epsilon}^c \cup \partial\Omega_{2,-\epsilon}^c} G(\mathbf{x},\mathbf{x}')\nabla' v_{ext}(\mathbf{x}')\cdot\hat{n}'_{ext}dS' \tag{4.21}$$

Summing (4.20) with (4.21), integrations along the shared facet cancels as $\hat{n}'_{ext}$ points in opposite directions, figure 4.2, only the integrals along the internal facet, designated as $\partial\Omega_1 \cup \partial\Omega_2 = \partial\Omega$, and the outer facet of $\partial\Omega^c$ are left.

Now extending the outer boundary of $\partial\Omega^c$ up to infinity, since $v(\mathbf{x}) \to 0$ and $\nabla v(\mathbf{x}) \to 0$, then only integrals along $\partial\Omega = \partial\Omega_\epsilon \cup \partial\Omega_{-\epsilon}$ remain

$$\lim_{\partial\Omega^c\to\infty} v_{ext}(\mathbf{x}) = -\int_{\partial\Omega_\epsilon\cup\partial\Omega_{-\epsilon}} v_{ext}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'_{ext}dS' + \int_{\partial\Omega_\epsilon\cup\partial\Omega_{-\epsilon}} G(\mathbf{x},\mathbf{x}')\nabla' v_{ext}(\mathbf{x}')\cdot\hat{n}'_{ext}dS' \tag{4.22}$$

Once more we have four integrals similar to those already computed for Ω, (4.16) and (4.18), the first will give us

$$\lim_{\epsilon\to 0}\int_{\partial\Omega_\epsilon} v_{ext}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'_{ext}dS' = -\gamma_\cup v_{ext}(\mathbf{x}) \tag{4.23}$$

where $4\pi\gamma_\cup$ is the solid angle complementary to $4\pi\gamma_\cap$ obtained previously, both summed they complete the surface area of unit sphere centered at $\mathbf{x}$, implying $\gamma_\cup + \gamma_\cap = 1$.

The second integral along $\partial\Omega_\epsilon$ in (4.21) is analogous to (4.18). Finally both integrals along $\partial\Omega_{-\epsilon}$ reduce to integrals along $\partial\Omega$ as $\epsilon \to 0$. Observing that $\hat{n}'_{ext} = -\hat{n}$ for the region of $\partial\Omega^e_{1,-\epsilon}$ coincident with the original $\partial\Omega$ as shown by comparing figures 4.1-I and II, we substitute $\hat{n}'_{ext} = -\hat{n}'$ for $\mathbf{x}' \in \partial\Omega$, (4.22) is reformulated as

$$v_{ext}(\mathbf{x})(1-\gamma_\cup) = +\int_{\partial\Omega} v_{ext}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' - \int_{\partial\Omega} G(\mathbf{x},\mathbf{x}')\nabla' v_{ext}(\mathbf{x}')\cdot\hat{n}'dS' \tag{4.24}$$

This expression is analogous to (4.19), the integrals signs are flipped and $\gamma_\cap$ is substituted by $\gamma_\cup$.

We now sum (4.19) with (4.24), since the derivative is continuous, (4.4), the second integrals cancel, reformulating with $v_{int}(\mathbf{x})$ on the LHS we have

$$v_{int}(\mathbf{x}) = \int_{\partial\Omega} u_{int}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' - \gamma_\cap(\mathbf{x})u_{int}(\mathbf{x}) \tag{4.25}$$

$$= \int_{\partial\Omega} u_{int}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' + (\gamma_\cup(\mathbf{x})-1)u_{int}(\mathbf{x}) \tag{4.26}$$

Finally we can take the last step of our strategy and make $\phi(\mathbf{x})$ dependent on $u_{int}(\mathbf{x})$ by substituting this relation on the potentials split we initially made

$$\phi(\mathbf{x})_{int} = u_{int}(\mathbf{x}) + v_{int}(\mathbf{x}) \tag{4.27}$$

$$= \int_{\partial\Omega} u_{int}(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' + \gamma_\cup(\mathbf{x})u_{int}(\mathbf{x}) \tag{4.28}$$

While initially in order to solve the Poisson problem for ϕ with asymptotic boundary conditions with FEM we needed to bound the surroundings and mesh $\Omega \cup \Omega^c$, now with equation (4.28) we can solve Poisson's equation for u meshing only the system Ω, from these we have the values of u_{int} that we use to compute the boundary values of ϕ_{int}. We no longer need to mesh the surroundings Ω^c, because the original Poisson's problem for ϕ has now Dirichlet boundary conditions, this greatly reduces the computation cost [30].

4.3 Discreet relation between ϕ and u

To obtain the matrix form of (4.28) we can discretize the functions $\phi(\mathbf{x})$ and $u(\mathbf{x})$ using shape functions (3.27) and (3.28) from previous Chapter we get

$$\sum_{e=0}^{ne-1}\sum_{\alpha=0}^{3}\phi^e_\alpha N^e_\alpha(\mathbf{x}) = -\sum_{e=0}^{ne-1}\sum_{\alpha=0}^{3}u^e_\alpha\int_{\partial\Omega} N^e_\alpha(\mathbf{x}')\nabla' G(\mathbf{x},\mathbf{x}')\cdot\hat{n}'dS' + \gamma_\cup(\mathbf{x})\sum_{e=0}^{ne-1}\sum_{\alpha=0}^{3}u^e_\alpha N^e_\alpha(\mathbf{x}) \tag{4.29}$$

where we dropped the (int) notation and assume the mapping is between the internal values of $\phi(\mathbf{x})$ and $u(\mathbf{x})$ evaluated at $\mathbf{x}$ belonging on the surface of the system Ω. Choosing a particular $\mathbf{x}_i$ the LHS becomes ϕ_i since only one of the shape functions incident on vertex i is nonzero, the others will have a domain that will not contain $\mathbf{x}_i$ as nonzero (*cf* end of section 3.3).

Also, on the RHS the terms of double sum that are integrations along a shared tetrahedra facets will cancel because $\hat{n}'$ will point in opposite directions, so only integrations on the

triangles on the surface will be nonzero. Defining an external facet of a tetrahedral element close to the surface, $\partial\Omega^c$, figure 3.1, as Γ^e_{ext}, (4.29) is reformulated as

$$\phi_i = -\sum_{\{(e,\alpha)|n(e,\alpha)\in\partial\Omega\}} u^e_\alpha \int_{\Gamma^e_{ext}} N^e_\alpha(\mathbf{x}')\nabla' G(\mathbf{x}_i,\mathbf{x}')\cdot\hat{n}' dS' + \gamma_\cup(\mathbf{x}_i)u_i \tag{4.30}$$

This relation can be represented in matrix form as $\boldsymbol{\phi} = \mathrm{G}\mathbf{u}$, where both vectors have as many entries as there are vertices within Ω. A given i will identify a row of G, and all the integrals that are coefficients of a given u_j will be a part of the entry $\mathrm{G}_{i,j}$. If $i = j$ then in addition we have $\gamma_\cup(\mathbf{x}_i)$:

$$\mathrm{G}_{i,j} = -\sum_{\{(e,\alpha)|n(e,\alpha)=j\}} \int_{\Gamma^e_{ext}} N^e_\alpha(\mathbf{x}')\nabla' G(\mathbf{x}_i,\mathbf{x}')\cdot\hat{n}' dS' + \gamma_\cup(\mathbf{x}_i)\delta_{i,j} \tag{4.31}$$

Its our goal now to compute G.

4.4 Numerical calculation of the integrals and $\mathrm{G}_{i,j}$

Given the elements are tetrahedra, the domain of integration, Γ^e_{ext} in (4.31) is a collection of triangles with different orientations, $\hat{n}'$. We can build a single triangle in $d = 2$ and map it to each one of these triangles embedded in $d = 3$ as in figure 4.3.

Choosing N^2 points with coordinates $(\xi,\eta)^\intercal$ within this triangle, each one is associated to a given $\mathbf{x}' \in \Gamma^e_{ext}$. The integrand of (4.31) is then evaluated at these locations and multiplied by a weight, the Gaussian weight, and summed, giving us an approximation to the integral. Since Gaussian weights are defined for $d = 1$, further transformations are necessary. The figure below shows the mappings we will introduce. After relating a triangle embedded in $d = 3$ with a reference triangle. We will relate the latter with a centered square as in [32].

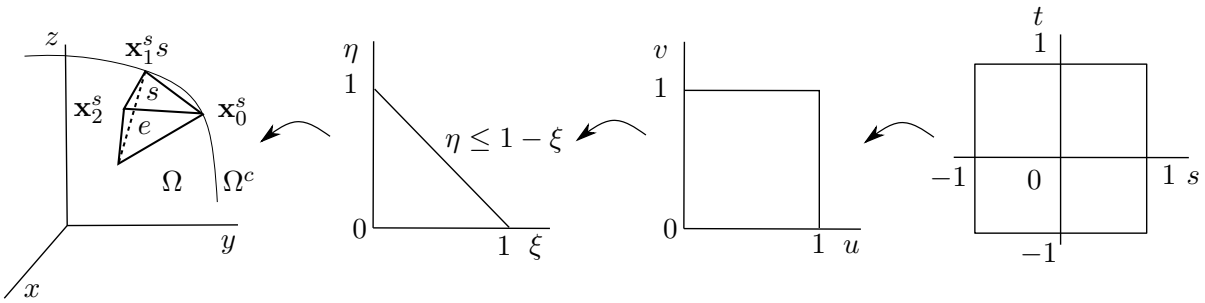

Figure 4.3: Sequence of coordinate transformations used to compute the integral $\mathrm{G}_{i,j}$.

4.4.1 Mapping

Mapping a point from the reference triangle to $(x^s, y^s, z^s)^T \in \Gamma^e_{ext}$ can be achieved using shape functions, $N^s_\alpha(\mathbf{x})$, to represent this location . These functions can be obtained from $N^e_\alpha(\mathbf{x})$ if

$n(s,\alpha) = n(e,\alpha) = i$ and $\mathbf{x}^s \in \Gamma^e_{ext}$.

$$\begin{aligned} x^s =& x^s_0 N^s_0(\mathbf{x}) + x^s_1 N^s_1(\mathbf{x}) + x^s_2 N^s_2(\mathbf{x}) \\ y^s =& y^s_0 N^s_0(\mathbf{x}) + y^s_1 N^s_1(\mathbf{x}) + y^s_2 N^s_2(\mathbf{x}) \\ z^s =& z^s_0 N^s_0(\mathbf{x}) + z^s_1 N^s_1(\mathbf{x}) + z^s_2 N^s_2(\mathbf{x}) \end{aligned} \tag{4.32}$$

These three equations are an identity operation, if we give $\mathbf{x}$ belonging to an element s as an input then the output is still $\mathbf{x}$. Not all three shape functions are independent, noting that for every $\mathbf{x} \in \Omega^e$ we have $N^e_0(\mathbf{x}) + N^e_1(\mathbf{x}) + N^e_2(\mathbf{x}) + N^e_3(\mathbf{x}) = 1$ then in particular its also true when $\mathbf{x} \in \Gamma^e_{ext}$. In this case one of the four shape functions will be zero, hence

$$\begin{aligned} & N^e_0(\mathbf{x}) + N^e_1(\mathbf{x}) + N^e_2(\mathbf{x}) = 1 \Leftrightarrow \\ & N^e_2(\mathbf{x}) = 1 - N^e_0(\mathbf{x}) - N^e_1(\mathbf{x}) \Leftrightarrow \\ & N^s_2(\mathbf{x}) = 1 - N^s_0(\mathbf{x}) - N^s_1(\mathbf{x}) \end{aligned} \tag{4.33}$$

Substitution of (4.33) into (4.32) gives us

$$\begin{aligned} x^s =& (x^s_0 - x^s_2) N^s_0(\mathbf{x}) + (x^s_1 - x^s_2) N^s_1(\mathbf{x}) + x^s_2 \\ y^s =& (y^s_0 - y^s_2) N^s_0(\mathbf{x}) + (y^s_1 - y^s_2) N^s_1(\mathbf{x}) + y^s_2 \\ z^s =& (z^s_0 - z^s_2) N^s_0(\mathbf{x}) + (z^s_1 - z^s_2) N^s_1(\mathbf{x}) + z^s_2 \end{aligned} \tag{4.34}$$

Using now the shape functions instead as parameters we define

$$N^s_\alpha = \begin{cases} \xi & \text{if } \alpha = 0 \\ \eta & \text{if } \alpha = 1 \\ 1 - \xi - \eta & \text{if } \alpha = 2 \end{cases} \tag{4.35}$$

The reference triangle in figure 4.3 is bounded by $\eta \leq 1 - \xi$ as shown by noting that $0 \leq N^e_{0,1,2} \leq 1$ implies

$$\begin{aligned} & 0 \leq 1 - N^s_0(\mathbf{x}) - N^s_1(\mathbf{x}) \leq 1 \Leftrightarrow \\ & N^s_1(\mathbf{x}) \leq 1 - N^s_0(\mathbf{x}) \Leftrightarrow \\ & \eta \leq 1 - \xi \end{aligned} \tag{4.36}$$

This triangle is mapped by the system (4.34), in matrix form we have

$$\begin{pmatrix} x^s \\ y^s \\ z^s \end{pmatrix} = \begin{pmatrix} x^s_2 & x^s_0 - x^s_2 & x^s_1 - x^s_2 \\ y^s_2 & y^s_0 - y^s_2 & y^s_1 - y^s_2 \\ z^s_2 & z^s_0 - z^s_2 & z^s_1 - z^s_2 \end{pmatrix} \begin{pmatrix} 1 \\ \xi \\ \eta \end{pmatrix} \Leftrightarrow \mathbf{x}^s = T^s \boldsymbol{\xi} \tag{4.37}$$

While $\mathbf{x}^s$ belongs to a triangular element embedded in $d = 3$ with vertices $\mathbf{x}^s_0$, $\mathbf{x}^s_1$ and $\mathbf{x}^s_2$, $(\xi, \eta)^\intercal$ belong to a triangular element defined by $\eta \leq 1 - \xi$ with $0 \leq \xi \leq 1$ as obtained in (4.36). To each point in the last one is mapped into one of the first. When $\xi = 1$ and $\eta = 0$ then this point inside the reference triangle is associated with $\mathbf{x}^s_0$, that is the vertex where

$N_0^e = 1$. In analogous way $\xi = 0$ and $\eta = 1$ will be associated to $\mathbf{x}_1^s$ and when both are zero to $\mathbf{x}_2^s$, where $N_2^e = 1 - \xi - \eta$.

If we know how to map points we can map variations. When ξ varies while η is constant results in $(d\mathbf{x}^s)_\eta$. If instead we vary η with ξ fixed we have $(d\mathbf{x}^s)_\xi$. Both are given by:

$$(d\mathbf{x}^s)_\eta = \begin{pmatrix} x_0^s - x_2^s \\ y_0^s - y_2^s \\ z_0^s - z_2^s \end{pmatrix} d\xi \quad \text{and} \quad (d\mathbf{x}^s)_\xi = \begin{pmatrix} x_1^s - x_2^s \\ y_1^s - y_2^s \\ z_1^s - z_2^s \end{pmatrix} d\eta \tag{4.38}$$

With this two vectors we can define a parallelogram. With it, areas are then mapped as

$$\begin{aligned} dS &= |(d\mathbf{x}^s)_\eta \times (d\mathbf{x}^s)_\xi| \\ &= ((y_0^s z_1^s - z_0^s y_1^s)^2 + (x_0^s z_1^s - z_0^s x_1^s)^2 + (x_0^s y_1^s - y_0^s x_1^s)^2)^{1/2} d\xi d\eta \\ &= J d\xi d\eta \end{aligned} \tag{4.39}$$

Mapping from a reference triangle to a reference square is represented in figure 4.3. As described in [32] first transform the triangle into a square using

$$\begin{aligned} \xi &= u \\ \eta &= (1-u)v \end{aligned} \tag{4.40}$$

where $0 \leq u, v \leq 1$. Variations are given by

$$\begin{pmatrix} d\xi \\ d\eta \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -v & 1-u \end{pmatrix} \begin{pmatrix} du \\ dv \end{pmatrix} \tag{4.41}$$

and in analogous way areas are mapped by

$$dS = (1-u) du dv \tag{4.42}$$

Finally we can relate this square to a centered and rescaled one using the transformation

$$\begin{aligned} u &= \frac{1+s}{2} \\ v &= \frac{1+t}{2} \end{aligned} \tag{4.43}$$

with $-1 \leq s, t \leq 1$, final transformation in figure 4.3. Variations are then as

$$\begin{pmatrix} du \\ dv \end{pmatrix} = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} ds \\ dt \end{pmatrix} \tag{4.44}$$

and areas are

$$dS = \frac{1}{4} ds dt \tag{4.45}$$

With these transformations, a generic integral along a reference triangle such as

$$I = \int_0^1 d\xi \int_0^{1-\xi} f(\xi, \eta) d\eta \tag{4.46}$$

can be reformulated into

$$I = \int_{-1}^{1} \int_{-1}^{1} f(u, (1-u)v)(1-u) du dv \tag{4.47}$$

$$= \int_{-1}^{1} \int_{-1}^{1} f\left(\frac{1+s}{2}, \frac{(1-s)(1+t)}{4}\right) \frac{1-s}{8} ds dt \tag{4.48}$$

A Gaussian quadrature as introduced in [12] can now be used, the approximation is done by evaluating $f(s,t)$ at each one of the N^2 points within the square, multiplied by the Jacobian $\frac{1-s_i}{8}$ and Gaussian weight $w_i w_j$.

$$I \approx \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} f\left(\frac{1-s_i}{2}, \frac{(1-s_i)(1-t_i)}{4}\right) \frac{1-s_i}{8} w_i w_j \tag{4.49}$$

Introducing the transformation (4.37) and (4.39) into the integral in (4.31) as well as replacing N_α^e by the definition (4.35) we arrive at

$$-\int_0^1 \int_0^{1-\xi} N_\alpha^s \hat{n}(\mathrm{T}^s \boldsymbol{\xi}) \cdot \frac{(T^s \boldsymbol{\xi} - \mathbf{x}_i)}{4\pi |T^s \boldsymbol{\xi} - \mathbf{x}_i|^3} J^s d\xi d\eta \tag{4.50}$$

where the minus sign comes from the gradient of the Green function. This integral is of the form (4.46) and can now be approximated analogously substituting into (4.49)

$$f(\xi, \eta) = -N_\alpha^s \hat{n}(T^s \boldsymbol{\xi}) \cdot \frac{(T^s \boldsymbol{\xi} - \mathbf{x}_i)}{4\pi |(T^s \boldsymbol{\xi} - \mathbf{x}_i)|^3} J^s \tag{4.51}$$

with

$$\begin{aligned} \xi &= \frac{1+s}{2} \\ \eta &= \frac{(1-s)(1+t)}{4} \end{aligned} \tag{4.52}$$

plugged into $\boldsymbol{\xi}$.

The next section will show the algorithm to compute these integrals and the $\mathrm{G}_{i,j}$ entries as in (4.31).

4.5 Algorithm for numerical integration and computation of G

The function `boundary_matrix_G` in Appendix H will build matrix entries $\mathrm{G}_{i,j}$ as described by (4.31) and the corresponding indices i and j. A given i is chosen and will identify a row of the matrix. We then choose a triangular element from the surface s and by running through

each $\alpha \in \{0,1,2\}$ an integral is approximated with (4.49) using N^2 Gaussian points. These calculations for a given i are stored as well as the corresponding indices i and $j = n(s,\alpha)$ in three separate lists. With these three, the summations in (4.31) are made by `csr_matrix` for the same $G_{i,j}$ entry.

The function is composed by four steps that we will describe below:

1. Setup Gaussian locations and weights and the Jacobians

Initially the locations and weights for the reference triangle are built with the function `gauss_triangle`, where `Ngauss` is given as an input and specifies the number of points along one dimension, the function will build `Ngauss**2` points locations (ξ, η) within the reference triangle listed in `xieta` and the corresponding Gaussian `weights`.

Then the Jacobians for the transformations given in (4.39) are computed from the locations of vertices on the boundary picked from `ptot` by the indices in `alphas`. Finally the indice lists `Ig`, `Jg` for the `G` entries start empty.

```
epsilon=10**(-15)                       #Tolerance to detect zero integrals.
xi_eta,weights=gauss_triangle(Ngauss)#Ref triangle Gauss locs and weights.
Jacobians=fJ(alphas,ptot)               #Jacobians from ref triangles.
Ig,Jg,G=[],[],[]                        #Where future matrix entries and
                                        #indices will be kept.
```

2. Pick a vertex from the boundary and an element s and test whether the integral 4.50 is zero

A vertex from the surface with index `i` and location `x_i` is chosen. A file is initialized on the first iteration `k=0`, or the current indices and matrix entries are saved to prevent these lists from getting too large as the `for` cycles advance. Since in (4.31), $(\mathbf{x}'(\xi,\eta) - \mathbf{x}_i)\cdot\hat{n}'$ gives the same value for every $\mathbf{x}' \in \Gamma^e_{ext}$ we choose in particular $(\mathbf{x}^s_0 - \mathbf{x}_i)\cdot\hat{n}'$ giving us `isnormal`, it will be used in computing the integrand `f` later but we can use its value now to decide whether the integral will be zero or not. Given that the system we will use is a box, many integrals will be zero. If `isnormal` is zero then further calculations are skipped by the `continue` preventing unnecessary computation.

```
for k,i in enumerate(Igs):  #Pick a vertex from the surface with index i.
"[Code Block]: Initialize a file or save"
    x_i = ptot[:,i]         #Location of vertex i.
    for s in range(alphas.shape[1]): #Pick a triangle element.
        ig,jg,kg = alphas[:,s]  #The 3 vertices locations for s.
        x0s = ptot[:,ig]
        x1s = ptot[:,jg]
        x2s = ptot[:,kg]
        n = normals[:,s]    #The normal to element s.
        d=x0s-x_i           #The distance from x_i to element s.
        isnormal=dot(n,d)   #If isnormal=0, skip further calculations.
```

```
        if abs(isnormal) < epsilon:
            continue
```

3. Compute the integrals

If it is not zero then the transformation matrix in (4.37) is computed from the element `s` vertices `x0s`, `x1s` and `x2s`. The integrals for each `a` are computed by summing the integrand `f` for `xi` and `eta` along the reference triangle which were computed in step **1**. The global indices `i` and `j` are then saved after being fixed by the block structure of the Jacobian, `i*5`,`j*5` gives the 5×5 blocks locations in the Jacobian $5nq \times 5nq$, further addition of `+3` along the columns and `+4` along the rows to give the final `G` entry location.

```
x0,y0,z0 = x0s                  #Build the coordinate transformation matrix T
x1,y1,z1 = x1s                  #from ref triangle to triangle element s.
x2,y2,z2 = x2s
T = array([[x2,x0-x2,x1-x2],
           [y2,y0-y2,y1-y2],
           [z2,z0-z2,z1-z2]])
Js = Jacobians[s]               #Pick the corresponding Jacobian.
for a in range(3):              #For a given Ns_alpha compute the integral, I.
    I=0
    for w,(xi,eta) in zip(weights,xi_eta):
        x = dot(T,array([1,xi,eta]))
        x = x-x_i
        x3 = 4*pi*norm(x)**3
        f = Ns_a(a,xi,eta)*isnormal*Js/x3
        I+=w*f
    G.append(-I)                #(-1) from derivative of Green fc.
    j = alphas[a][s]            #Append Gij matrix indices fixed
    Ig.append(i*5+3)            #by the Jacobian block structure.
    Jg.append(j*5+4)
```

4. Compute the diagonal terms $\gamma_\cup$

Finally $\gamma_\cup$ terms are computed for all boundary vertices `i`. The Kronecker Delta term in (4.31) implies that we add these term only to diagonal $5 * 5$ blocks which are then fixed by `+3` and `+4` as in step **3**.

```
for i in Igs:
    x_i = ptot[:,i]
    G.append(fgamma(x_i,pbox))
    Ig.append(i*5+3)
    Jg.append(i*5+4)
"[Block of code]: Save diagonal entries"
```

The final sum of all contributions for each $G_{i,j}$ is made by `csr_matrix` which will sum all those entries in G list that have the same coordinates in `IgG` and `JgG`.

4.6 Restatement of Poisson's problem

The original statements that determine ϕ are now replaced by

1. Using FEM, find u such that

$$\begin{cases} \nabla^2 u = \nabla \cdot \mathbf{m} & \text{if } \mathbf{x} \in \Omega \\ u = 0 & \text{if } \mathbf{x} \in \Omega^c \\ \frac{\partial u_{int}}{\partial n} = \mathbf{m} \cdot \hat{n} & \text{if } \mathbf{x} \in \partial\Omega \end{cases} \tag{4.53}$$

2. Map u_{int} into ϕ_{int} using

$$\boldsymbol{\phi}_{int} = \mathrm{G}\mathbf{u}_{int} \tag{4.54}$$

3. Using FEM, find ϕ such that

$$\begin{cases} \nabla^2 \phi = \nabla \cdot \mathbf{m} & \text{if } \mathbf{x} \in \Omega \\ \phi(\mathbf{x}) = \phi_{int} & \text{if } \mathbf{x} \in \partial\Omega \end{cases} \tag{4.55}$$

For the ϕ residual we have

$$s^e_{\phi,\alpha} = \int_{\Omega^e} \nabla \cdot \mathbf{m} N^e_\alpha(\mathbf{x}) d^3x + \int_{\Omega^e} \nabla\phi \cdot \nabla N^e_\alpha(\mathbf{x}) d^3x \tag{4.56}$$

where we do not have the boundary integral since we imposed Dirishlet Boundary conditions on $\partial\Omega$ [21, p. 92, 108]. Instead of (3.36) we only have now the first two integrals

$$s^e_{\phi,\alpha} = s^{src,e}_{\phi,\alpha} + s^{stf,e}_{\phi,\alpha} \tag{4.57}$$

The first is given by (3.38) and the second is (3.41) with the IMR substituted. This expression is only valid for vertices inside Ω, that is $n(e,\alpha) \in \Omega \setminus \partial\Omega$.

The two Jacobians $\frac{\partial s^e_\alpha}{\partial \phi^e_\gamma}$ and $\frac{\partial s^e_\alpha}{\partial \mathbf{m}^e_\gamma}$ are now only given by the first terms of (3.57) and (3.56), since the second ones are related to the boundary integral.

$$\frac{\partial s^e_{\phi,\alpha}}{\partial \phi^e_\gamma} = -\frac{1}{2} K^e_{\alpha,\beta} \tag{4.58}$$

$$Q^e_{\alpha\gamma} = \frac{\partial s^e_{\phi,\alpha}}{\partial \mathbf{m}^e_\gamma} = -\frac{\Omega^e}{8} \nabla N^e_\gamma \tag{4.59}$$

From the mapping $\boldsymbol{\phi}_{int} = \mathrm{G}\mathbf{u}_{int}$ for the boundary vertices we can build the condition $0 = \mathrm{G}\mathbf{u}_{int} - \boldsymbol{\phi}_{int}$ that also has to be verified just like the LLG and the two Poisson's equations.

To evaluate how the boundary values of $\mathbf{u}_{int}$ and $\boldsymbol{\phi}_{int}$ satisfy this condition we define a new residual just for the boundary vertices

$$\mathbf{s}^{\partial}_{\phi} = \mathrm{G}\mathbf{u}_{int} - \boldsymbol{\phi}_{int} \tag{4.60}$$

each component of these vector is given by

$$\text{with IMR:} \quad s^{\partial}_{\phi,i} = \sum_{j \in \partial\Omega} \mathrm{G}_{i,j} \frac{u_j + u_{0,j}}{2} - \frac{\phi_i + \phi_{0,i}}{2} \quad \text{for } i \in \partial\Omega \tag{4.61}$$

where we introduced IMR. This is the residual at boundary vertices instead of (4.57), hence instead of having the entry (4.58) and (4.59) we have the Jacobians

$$\frac{\partial s^{\partial}_{\phi,i}}{\partial u_k} = \frac{\mathrm{G}_{i,k}}{2} \tag{4.62}$$

$$\frac{\partial s^{\partial}_{\phi,i}}{\partial \phi_k} = -\frac{1}{2} \tag{4.63}$$

which are easily obtained since (4.60) is linear and where $i, k \in \partial\Omega$. As in the previous Chapter we define the weak form for the residual of u similar to (3.32)

$$s^{e}_{u,\alpha} = \int_{\Omega^e} \nabla \cdot \mathbf{m} N^{e}_{\alpha}(\mathbf{x}) d^3x + \int_{\Omega^e} \nabla u \cdot \nabla N^{e}_{\alpha}(\mathbf{x}) d^3x - \int_{\partial\Omega^e} N^{e}_{\alpha}(\mathbf{x}) \nabla u \cdot \hat{n} dS \tag{4.64}$$

which is split in three parts, the source term is the same as (3.40), the stiffness term is also the same as in (3.43) and the surface integral the same as in (3.55) which was obtained from the discontinuity of the derivatives in in (3.49). The difference now is that the second integral in (3.48) does not exist as we are on the internal boundary and in the first integral is substituted the Neumann boundary condition in (4.53) therefore yielding (3.49) as before. As a consequence the Jacobians for $s^{e}_{u,\alpha}$ are given by the two terms in (3.56) and only the first term in (3.57).

Now we have all the residuals necessary to build the system of equations for NR, that is the task for next Chapter.

Chapter 5

Residual Vector and Jacobian Matrix Assembly

In Chapter 3 we derived expressions for the linear approximations of the residual of LLG and Poisson's equation $\mathbf{r}^e_\alpha$ and $s^e_{\phi,\alpha}$. In Chapter 4 we modified Poisson's problem to use only information from the system by introducing a new potential and a new residual $s^e_{u,\alpha}$. Each one of these three residuals can now be specified for each vertex i yielding linear equations.

In this Chapter we will show how to organize this set of equations and develop a Python code to build it. Of all contributions, we will focus on the two Poisson's Equations with BEM coupling and the exchange term as they will encompass all the main algorithm strategies, the other terms are given in Appendix.

We then introduce our implementation of Newton-Raphson method and briefly refer to the algorithms of GMRES with the ILU and Algebraic Multigrid preconditioners to solve the linear system of equations.

5.1 A larger Jacobian, J

The configuration for every vertex in the system Ω is specified by the magnetization $\mathbf{m}_i$ and two potentials ϕ_i and u_i which can be assembled into a vector with five entries $\vec{x} = (m_x, m_y, m_z, \phi, u)^T_i$. To measure how close a local configuration composed by the vertex i and its connected neighbours is to the exact solutions of LLG equation and Poisson's equations for both potentials, we define at i a five entry residual vector $\vec{r}_i = (r_x, r_y, r_z, s_\phi, s_u)^T_i$.

If we have a particular configuration for i and his j neighbours, if we change the configuration for each j, how does the residual for i change? Consider for example the first order approximation of the discretized LLG residual. When only the magnetization $\mathbf{m}_j$ varies from a particular configuration, $\mathbf{m}_{j,p}$, the variation of the residual at i is obtained from the 3×3 Jacobian matrices of first derivatives and the following linear approximation

$$\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}_i = \begin{pmatrix} r_{x,p} \\ r_{y,p} \\ r_{z,p} \end{pmatrix}_i + \begin{pmatrix} \frac{\partial r_x}{\partial m_x} & \frac{\partial r_x}{\partial m_y} & \frac{\partial r_x}{\partial m_z} \\ \frac{\partial r_y}{\partial m_x} & \frac{\partial r_y}{\partial m_y} & \frac{\partial r_y}{\partial m_z} \\ \frac{\partial r_z}{\partial m_x} & \frac{\partial r_z}{\partial m_y} & \frac{\partial r_z}{\partial m_z} \end{pmatrix}_{i,i} \begin{pmatrix} \Delta m_x \\ \Delta m_y \\ \Delta m_z \end{pmatrix}_i + \cdots + \begin{pmatrix} \frac{\partial r_x}{\partial m_x} & \frac{\partial r_x}{\partial m_y} & \frac{\partial r_x}{\partial m_z} \\ \frac{\partial r_y}{\partial m_x} & \frac{\partial r_y}{\partial m_y} & \frac{\partial r_y}{\partial m_z} \\ \frac{\partial r_z}{\partial m_x} & \frac{\partial r_z}{\partial m_y} & \frac{\partial r_z}{\partial m_z} \end{pmatrix}_{i,j} \begin{pmatrix} \Delta m_x \\ \Delta m_y \\ \Delta m_z \end{pmatrix}_j + \cdots \tag{5.1}$$

These two matrices shown represent the relation between the residual at vertex i and a variation in magnetization at i itself and at a neighbour j, their expressions were obtained in previous chapters for each term of LLG residual when two vertices are connected. When ϕ_i varies as well an additional vector term is added to (5.1) given by $(\frac{\partial r_x}{\partial \phi}, \frac{\partial r_y}{\partial \phi}, \frac{\partial r_z}{\partial \phi})^T_{i,j} \Delta\phi_i$. If there is no edge in the mesh connecting (i, j), this matrix and vector are made out of zeros.

An analogous expression for the linear dependence of $s_{\phi,i}$ or $s_{u,i}$ with magnetization and potential at j is given by

$$s_{\phi,i} = s_{\phi,i,p} + \cdots + \begin{pmatrix} \frac{\partial s_{\phi,i}}{\partial m_{x,j}} & \frac{\partial s_{\phi,i}}{\partial m_{y,j}} & \frac{\partial s_{\phi,i}}{\partial m_{z,j}} \end{pmatrix}_{i,j} \begin{pmatrix} \Delta m_x \\ \Delta m_y \\ \Delta m_z \end{pmatrix}_j + \frac{\partial s_{\phi,i}}{\partial \phi_j} \Delta\phi_j + \dots \tag{5.2}$$

except when $i \in \partial\Omega$ where $s_{\phi,i}$ has the form (4.61).

We can organize the dependence of the four residuals with the variation of a configuration of a given j neighbour as

$$\begin{pmatrix} r_x \\ r_y \\ r_z \\ s_\phi \\ s_u \end{pmatrix}_i = \begin{pmatrix} r_{x,p} \\ r_{y,p} \\ r_{z,p} \\ s_{\phi,p} \\ s_{u,p} \end{pmatrix}_i + \left(\begin{array}{c|c|c} & & 0 \\ \frac{\partial \mathbf{r}}{\partial \mathbf{m}} & \frac{\partial \mathbf{r}}{\partial \phi} & 0 \\ & & 0 \\ \hline \frac{\partial s_\phi}{\partial \mathbf{m}} & \frac{\partial s_\phi}{\partial \phi} & 0 \\ \hline \frac{\partial s_u}{\partial \mathbf{m}} & 0 & \frac{\partial s_u}{\partial \phi} \end{array}\right)_{i,j} \begin{pmatrix} \Delta m_x \\ \Delta m_y \\ \Delta m_z \\ \Delta\phi \\ \Delta u \end{pmatrix}_j \tag{5.3}$$

This 5×5 matrix defined as $\mathrm{J}_{i,j}$ is associated with a generic edge (i, j) of the mesh, it has several blocks. The $\frac{\partial \mathbf{r}}{\partial \mathbf{m}}$ is a 3×3 matrix as in (5.1) that is composed by the time derivative term $\mathrm{A}^e_{\alpha,\alpha}$ as in (3.64), the triple term contribution given by $\mathrm{D}^e_{\alpha,\alpha}$ as (3.76) and the exchange contribution $\mathrm{E}^e_{\alpha,\beta}$ given in (3.94). The 3×1 column vector $\frac{\partial \mathbf{r}}{\partial \phi}$ is $\mathrm{P}^e_{\alpha,\beta}$ in (3.77). The two 1×3 row vectors $\frac{\partial s_\phi}{\partial \mathbf{m}}$ and $\frac{\partial s_u}{\partial \mathbf{m}}$ both defined as $\mathrm{Q}^e_{\alpha,\beta}$ in (3.56). And the the two scalars $\frac{\partial s_\phi}{\partial \phi}$ and $\frac{\partial s_u}{\partial \phi}$ both derived in (3.57). The variation vector in (5.3) is defined as $\Delta\vec{x}_j$ and multiplies the matrix giving how much the residual changes when the particular configuration for the j vertex changes as well.

Since i has several neighbours we have to extend (5.3) to include their Jacobians. Therefore in an analogous way as we summed matrix vector multiplications in (5.1) we have

$$\vec{r}_i = \vec{r}_{i,p} + \mathrm{J}_{i,0}\Delta\vec{x}_0 + \cdots + \mathrm{J}_{i,j}\Delta\vec{x}_j + \cdots + \mathrm{J}_{i,n_q-1}\Delta\vec{x}_{n_q-1} \tag{5.4}$$

By joining all 5×5 blocks in (5.4) into a $5 \times 5n_q$ matrix and all n_q variations into a single $5n_q \times 1$ column vector this sum can be represented as a dot product of both, which represents all possible relations of i with all vertices in the mesh including vertices that are not connected. This situation is analogous to what we have did to obtain (3.45) but with a much larger matrix and vector.

We can go further now and establish the same relation (5.4) for all i vertices generating n_q matrices with shape $5 \times 5n_q$. Observe now that each one of these matrices is in fact a $5 \times 5n_q$ row block of a much larger Jacobian matrix $\mathbf{J}$ which results from assembling all these into a final $5n_q \times 5n_q$ Jacobian. By setting the LHS of equation (5.4) equal to zero for all i

and rearranging we arrive at

$$\left(\begin{array}{cccc} & & & \\ J_{0,0} & J_{0,1} & \dots & J_{0,n_q-1} \\ & & & \\ J_{1,0} & J_{1,1} & \dots & J_{1,n_q-1} \\ & & & \\ \vdots & \vdots & \ddots & \vdots \\ & & & \\ J_{n_q-1,0} & J_{n_q-1,1} & \dots & J_{n_q-1,n_q-1} \\ & & & \end{array}\right)\left(\begin{array}{c} \Delta\mathbf{m}_0 \\ \Delta\phi_0 \\ \Delta u_0 \\ \hline \Delta\mathbf{m}_1 \\ \Delta\phi_1 \\ \Delta u_1 \\ \hline \\ \vdots \\ \\ \hline \Delta\mathbf{m}_{n_q-1} \\ \Delta\phi_{n_q-1} \\ \Delta u_{n_q-1} \end{array}\right) = -\left(\begin{array}{c} \mathbf{r}_0 \\ s_{\phi,0} \\ s_{u,0} \\ \hline \mathbf{r}_1 \\ s_{\phi,1} \\ s_{u,1} \\ \hline \\ \vdots \\ \\ \hline \mathbf{r}_{n_q-1} \\ s_{\phi,n_q-1} \\ s_{u,n_q-1} \end{array}\right) \tag{5.5}$$

This is the linear system of equations that we have to solve iteratively as we described in Section 3.2. Remember that $\mathbf{J}$ on the LHS and the RHS residual, $\underline{\mathbf{r}}$, both depend on $\mathbf{m}_0$, ϕ_0 and u_0 and then are evaluated at a particular guessed configuration, that we have decided to start at $\mathbf{m}_0$, ϕ_0 and u_0. After assembling the $\mathbf{J}$ and $\underline{\mathbf{r}}$ we solve for the variations, $\Delta\underline{\mathbf{x}}$. We add this variation to our guess giving us the next particular configurations we use to recompute $\mathbf{J}$ and $\underline{\mathbf{r}}$ while maintaining the same $\mathbf{m}_0$, ϕ_0 and u_0. As we proceed, the residual will on the RHS get smaller progressively until a prescribed tolerance. The current particular configuration thereby obtained is close to the exact solutions, that is we have an approximation for the update of the initial magnetization and potentials that satisfies the LLG equation and both Poisson's equations. Repeating the whole process we construct the history of the system from the initial configuration.

Notice we have to assemble and recompute some of the blocks of this huge sparse matrix, $\mathbf{J}$, and vector, $\underline{\mathbf{r}}$, at every iteration. This has to be done quickly. In the next section we review the techniques proposed by [24] for the assembly of Poisson's stiffness matrix and we propose the extension of such techniques for all terms in the LLG equation.

5.2 Poisson's Contribution

To build the Poisson's stiffness matrix we have to return to the gradients of the basis functions and introduce the technique to compute them. Remember from (3.23) that for an element e the basis has the form

$$N_\alpha^e(\mathbf{x}) = \frac{1}{6\bar{\Omega}^e}(c_{\alpha,0}^e + c_{\alpha,1}^e x + c_{\alpha,2}^e y + c_{\alpha,3}^e z) \tag{5.6}$$

where $c_{\alpha,i}^e$ is the cofactor of entry (α, i) of the matrix X composed by the spatial locations for each vertex in a given element. Its gradient is given by

$$\nabla N_\alpha^e(\mathbf{x}) = \frac{1}{6\bar{\Omega}^e}\begin{pmatrix} c_{\alpha,1} \\ c_{\alpha,2} \\ c_{\alpha,3} \end{pmatrix} \tag{5.7}$$

and is independent of $\mathbf{x}$.

To build the stiffness matrix we need to compute these gradients for all elements and that in turn requires computing the three cofactors for each one. From (3.20) take for instance

$$c_{0,1} = -\left(\begin{vmatrix} y_2 & z_2 \\ y_3 & z_3 \end{vmatrix} - \begin{vmatrix} y_1 & z_1 \\ y_3 & z_3 \end{vmatrix} + \begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix} \right) \tag{5.8}$$

the minus sign outside the parenthesis is the signal of this cofactor. A simple algorithm to compute all gradients would be to pick an element then its four vertices' global indices on column e of the connectivity matrix, whose spatial locations are used as inputs for each one of the three cofactor expressions such as (5.8), giving us one gradient for a particular (e, α). Repeating the same process for all remaining α we get the four gradients of the four basis function of an element. These calculations are then carried on all other elements and saved.

A severe drawback for an algorithm like this in Python is the use of many `for` cycles, which are very slow [24]. In particular the issue is not mainly related with the Poisson's stiffness term for ϕ and u since it is built out of constant gradient terms and hence only needs to be computed once. The issue is when we have to go through the process of computing and assembling Jacobian blocks repeatedly, as occurs for all other terms of LLG which changes as magnetization and potentials evolve.

As [24] points out, computing the gradients and then the K entries in (3.44) can be reformulated with vectorial operations such as element-wise multiplications and sums, which by themselves are faster and reduce the number of required `for` cycles. The central idea is to vectorize all possible operations.

In the next section we will introduce these concepts for Poisson's stiffness matrix from [24] and then go further and extend them to develop the code for the **J** part associated with the Poisson's stiffness matrix for both potentials taking into account the BEM mapping between them, we build the three vector residuals associated with (3.36) for both potentials as well, that, once summed, constitute one of the parts of the final residual vector $\underline{\mathbf{r}}$ in (5.5).

5.2.1 From Gradients to Poisson's Stiffness Matrix

Our goal is to build the $n_q \times n_q$ matrix entries in (3.44) and then relate them to the **J** matrix. Start noticing that $c_{0,1}$ above can be reformulated. Temporarily complicating the expression we have

$$c_{0,1} = \begin{vmatrix} y_1 & z_1 \\ y_1 & z_1 \end{vmatrix} - \begin{vmatrix} y_2 & z_2 \\ y_3 & z_3 \end{vmatrix} + \begin{vmatrix} y_1 & z_1 \\ y_3 & z_3 \end{vmatrix} - \begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix} \tag{5.9}$$

Since determinants are linear row-wise we join the first with the last determinant and the middle ones

$$c_{0,1} = \begin{vmatrix} y_1 & z_1 \\ y_1 - y_2 & z_1 - z_2 \end{vmatrix} + \begin{vmatrix} y_1 - y_2 & z_1 - z_2 \\ y_3 & z_3 \end{vmatrix} \tag{5.10}$$

Multiplying the second by $(-1)(-1)$ yields

$$c_{0,1} = \begin{vmatrix} y_1 - y_3 & z_1 - z_3 \\ y_1 - y_2 & z_1 - z_2 \end{vmatrix} = (y_1 - y_3)(z_1 - z_2) - (y_1 - y_2)(z_1 - z_3) \tag{5.11}$$

Unlike the original expression (5.8) which had mixed components, the latter is a product of differences of the same component. This is important because it unlocks the possibility to use matrix element-wise operations provided we have the right data structures. Hence, we maintain the connectivity matrix as an $4 \times n_e$ array and we restructure the array of vertices locations from the shape $n_q \times 3$ to $3 \times n_q$.

Now, instead of computing all cofactors for a given element and proceeding with the next one as suggested in the simple algorithm, what we want is to compute the gradients for all vertices associated with a given α as a single vectorial operation. This is achieved as follows, we start by defining `alphae` as a Python array with the connectivity matrix entries, noticing that `alphae[a]` picks the `a` row, with n_e entries, which contains all global indices of all α vertices. From the array of locations `ptot`, all locations of all these vertices are sliced as `ptot[:,alphae[a]]`, which is a $3 \times n_e$ array where each column gives us the position in space of all vertices $i = n(e, \alpha = \texttt{a})$ for all e. These slices allow us to compute the required differences in (5.11), for example the first one, $y_1 - y_3$, can be computed for all e as `ptot[1,alphae[1]]-ptot[1,alphae[3]]` where the first index 1 indicates we are choosing row one of `ptot`, the row with all y components. Since we will ultimately require differences between all components we do better computing differences between vectors and later pick the right components. The following difference

$$\begin{pmatrix} x_\alpha^e \\ y_\alpha^e \\ z_\alpha^e \end{pmatrix} - \begin{pmatrix} x_\beta^e \\ y_\beta^e \\ z_\beta^e \end{pmatrix} \tag{5.12}$$

between the position of α vertices and the position of β vertices can be computed across all elements using

```
D_ab = ptot[:,alphae[a]]-ptot[:,alphae[b]]
```

Since (5.11) is composed of differences between components it can be computed for all elements using the arrays `D_12` and `D_13`. We pick the second row of `D_13`, the one with the differences $y_1^e - y_3^e$ for all elements, and the third row of `D_12`, which has the z component differences $z_1^e - z_2^e$. Both are then multiplied element-wise as `D_12[1,:]*D_13[2,:]` yielding $(y_1^e - y_3^e)(z_1^e - z_2^e)$ for all e, which is then summed to the second term in (5.11) as

```
(D_13[1,:]*D_12[2,:]-D_12[1,:]*D_13[2,:])*C
```

where the constant `C=1/(6*signedvol)` has all determinants of X for all elements element-wise multiplies every entry of the array in parenthesis.

The other two cofactor components for ∇N_0^e are the second and third entries of

$$\nabla N_0^e(\mathbf{x}) = \frac{1}{det(X)} \begin{pmatrix} (y_1 - y_3)(z_1 - z_2) - (y_1 - y_2)(z_1 - z_3) \\ (x_1 - x_2)(z_1 - z_3) - (x_1 - x_3)(z_1 - z_2) \\ (x_1 - x_3)(y_1 - y_2) - (x_1 - x_2)(y_1 - y_3) \end{pmatrix} \tag{5.13}$$

We observe that to assemble all gradients for $\alpha = 0$ can be done extending the element-wise operations we used for the first entry to the other two. Defining now a $3 \times 4n_e$ array of zeros

`Grads`, we can fill the first $3 \times n_e$ block with the gradients $\nabla N_0^e(\mathbf{x})$ as follows

```
Grads=zeros([3,4*ne]) #Initial G.
Grads[:,:ne] = array([(D_13[1,:]*D_12[2,:]-D_12[1,:]*D_13[2,:])*C,
                      (D_12[0,:]*D_13[2,:]-D_13[0,:]*D_12[2,:])*C,
                      (D_13[0,:]*D_12[1,:]-D_12[0,:]*D_13[1,:])*C])
```

The remaining three block with shape $3 \times n_e$ will receive the $\nabla N_1^e(\mathbf{x})$, $\nabla N_2^e(\mathbf{x})$ and $\nabla N_3^e(\mathbf{x})$ calculations

$$\texttt{Grads} = \left[\begin{array}{c|c|c|c} \alpha = 0 & \alpha = 1 & \alpha = 2 & \alpha = 3 \\ \texttt{[:ne]} & \texttt{[ne:2*ne]} & \texttt{[2*ne:3*ne]} & \texttt{[3*ne:]} \end{array}\right]_{3\times 4n_e} \tag{5.14}$$

has the Algorithm A.4 shows [24].

Once we have all entries for the matrix `Grads`, its structure allows fast computing of the stiffness matrix entries. Remember that each (i, j) entry of K matrix is given by

$$K_{ij} = \sum_{\{(e,\alpha,\beta)|i=n(e,\alpha)\wedge j=n(e,\beta)\}} \nabla N_\beta^e \cdot \nabla N_\alpha^e \Omega^e \tag{5.15}$$

We observe K_{ij} is a summation of dot products, $K^e_{\alpha,\beta}$, mapped by the connectivity matrix into the same (i, j). All entries of K then will require all $K^e_{\alpha,\beta}$ calculations, we start by organizing them as consecutive blocks of size n_e in the following `Kg` array

$$\texttt{Kg} = \left[\begin{array}{c|c|c|c|c|c|c} (0,0) & (0,1) & (0,2) & \dots & (\alpha,\beta) & \dots & (3,3) \\ \texttt{[:ne]} & \texttt{[ne:2ne]} & \texttt{[2*ne:3*ne]} & & \texttt{[c*ne:(c+1)*ne]} & & \texttt{[15*ne:]} \end{array}\right]_{1\times 16n_e} \tag{5.16}$$

where `c` stands for the index in $\{0, 1, ..., 15\}$ associated to a combination (α, β). Before handling the details of how to compute each entry in `Kg` we will focus now on its block structure to build the indices (i, j) for each entry. Defining `Ig` and `Jg` as two arrays of zeros with the same shape as `Kg` and the same 16 row block structure with length n_e

$$\texttt{Ig} = \left[\begin{array}{c|c|c|c|c|c|c} 0 & 0 & 0 & \dots & \alpha & \dots & 3 \end{array}\right]_{1\times 16n_e} \tag{5.17}$$

$$\texttt{Jg} = \left[\begin{array}{c|c|c|c|c|c|c} 0 & 1 & 2 & \dots & \beta & \dots & 3 \end{array}\right]_{1\times 16n_e} \tag{5.18}$$

for each `Kg` entry we assign for both the respective global index $i = n(e, \alpha)$ and $j = n(e, \beta)$. This is done with two simple arrays `ii` and `jj` [24], associated to α and β sequences in the n_e blocks of `Kg`

$$\texttt{ii} = \left[\begin{array}{c|c|c|c} 0,0,0,0, & 1,1,1,1, & 2,2,2,2, & 3,3,3,3 \end{array}\right]_{1\times 16} \tag{5.19}$$

$$\texttt{jj} = \left[\begin{array}{c|c|c|c} 0,1,2,3, & 0,1,2,3, & 0,1,2,3, & 0,1,2,3 \end{array}\right]_{1\times 16} \tag{5.20}$$

We cycle through all (α, β) combinations by ranging `c` from 0 to 15 and picking $\alpha =$ `ii[c]` and $\beta =$ `jj[c]`, then we fill each n_e block initially with zeros in `Ig` and `Jg` respectively with the n_e global indices associated to α and β, that is with `alphae[ii[c]]` and `alphae[jj[c]]`.

Once we have the three arrays, `Kg`, `Ig` and `Jg`, we give them as inputs to the `csr_matrix` Python function that builds the matrix and entries and if two or more entries have the same combination of (i, j), they are summed and assigned into that entry in the matrix. This function yielding an array in `csr` format and shape $n_q \times n_q$.

This array could be used in solving the Poisson's equation alone. However our goal is to make it a part of the larger Jacobian matrix. Observe that each $J_{i,j}$ block in $\mathbf{J}$ has the structure given in (5.3). The entry $K_{i,j}$ is equal to the two Poisson entries in the $J_{i,j}$ block in $\mathbf{J}$.

Where for the moment we ignore the fact that the boundary vertices have the residual given by (4.60). Observe that in $\mathbf{J}$ the global indices $5i$ and $5j$ locate the upperleft entry of $J_{i,j}$ 5×5 block. To locate the Poisson's ϕ entry in $\mathbf{J}$ we need to add a local increment, $(5i+3, 5j+3)$, for potential u we have $(5i+4, 5j+4)$. This observation amounts to multiplying `Ig` and `Jg` by 5 and then adding a local fixing `+3`. Therefore, to build the indices arrays that map the `Kg` $1 \times 16n_e$ entries into the Poisson entries in $\mathbf{J}$ we proceed as

```
Ig = zeros([16*ne,],dtype='int')#Initial Ig and Jg
Jg = zeros([16*ne,],dtype='int')
for c in range(4*4):              #Choose a combination (alpha,beta)
    Ig[c*ne:(c+1)*ne]=alphae[ii[c],:]*5+3 #(+4) if u
    Jg[c*ne:(c+1)*ne]=alphae[jj[c],:]*5+3 #(+4) if u
```

The details to finally handle each entry in each n_e block of `Kg` we will use a particular example. Choose the first two columns of `Grads`, each one is associated with element $e = 0$ for vertices $\alpha = 0$ and $\beta = 1$, the dot product is

$$\nabla N^0_{\beta=0} \cdot \nabla N^0_{\alpha=1} = \frac{1}{(6\bar{\Omega}^e)^2} \begin{pmatrix} c^0_{0,x} & c^0_{0,y} & c^0_{0,z} \end{pmatrix} \begin{pmatrix} c^0_{1,x} \\ c^0_{1,y} \\ c^0_{1,z} \end{pmatrix} = c^0_{0,x}c^0_{1,x} + c^0_{0,x}c^0_{1,y} + c^0_{0,z}c^0_{1,z} \tag{5.21}$$

A careful observation shows this operation is the same as multiplying both column vectors element-wise and then sum the vector column-wise. The structure of `Grads` is the one that allows a generalization, we element-wise multiply two $3 \times n_e$ blocks associated with α and β resulting still in a $3 \times n_e$ array, that once summed column-wise yields a row vector $1 \times n_e$ where each one of the entries is $\nabla N^e_\alpha \cdot \nabla N^e_\beta \Omega^e$.

Hence the first five (α, β) combinations that will be used in computing the stiffness matrix as in [24] are given by

```
Kg = zeros([ne*16,])
                                  #(alpha,beta)
Kg[0:ne]          = sum(G_0*G_0)*vol   #(0,0)
Kg[ne:2*ne]       = sum(G_0*G_1)*vol   #(0,1)
Kg[2*ne:3*ne]     = sum(G_0*G_2)*vol   #(0,2)
Kg[3*ne:4*ne]     = sum(G_0*G_3)*vol   #(0,3)
Kg[4*ne:5*ne]     = Kg[ne:2*ne]        #(1,0)
```

where we observe that since K is symmetric the $(0,1)$ block is the same as the $(1,0)$. All other α, β combinations are in Algorithm B.4.

Now that we have the necessary information to build both $\frac{\partial s_{\phi,i}}{\partial \phi_j}$ and $\frac{\partial s_{u,i}}{\partial u_j}$ for the (i,j) blocks of $\mathbf{J}$, using the `Kg` array together with indices `Ig` and `Jg` we have

```
Kgphi = csr_matrix((Kg,(Ig,Jg),shape=(5*nq,5*nq))
```

as a $5n_q \times 5n_q$ sparse matrix where only the $\frac{\partial s_{\phi,i}}{\partial \phi_j}$ entries within the 5×5 blocks associated with connected vertices are nonzero. In a similar manner, the process of assembling the stiffness matrix for the potential u is given by

```
Kgu = csr_matrix((Kg,(Ig+1,Jg+1),shape=(5*nq,5*nq))
```

since the entries are the same but one entry further down the diagonal of each 5×5 block.

However, remember that for the vertices that belong to $\partial\Omega$ we have the potential ϕ_i determined by all boundary values of the potential u_j and not an equation that related it with the neighbours represented by the coefficients at each row for all i within Ω. We have then to replace all entries in `Kgphi` at every row associated with $i \in \partial\Omega$ by (4.63) and (4.62). In practice we define first the `Igs` as an array of global indices of vertices at the boundary, second we assemble the array `G` introduced in section 4.5 that maps $\mathbf{u}$ into $\boldsymbol{\phi}$, then we introduce the Jacobians of (4.62) as

```
Kgphi[Igs*5+3,Igs*5+3]=-1
Kgphi+=G
```

where we postpone the introduction of the $\frac{1}{2}$ factor from IMR. By summing `Kgphi` and `Kgu` we will have a $5n_q \times 5n_q$ `csr` matrix that can later be added to other assembled Jacobian contributions.

An important observation from the two previous contributions is that they do not depend on the current configuration $\underline{\mathbf{x}}$, meaning it has only to be done once. In fact there are other contributions that are also constants as well. The time derivative term `A` in Algorithm F.1 and both `Q` for u and ϕ are also constant and described in Algorithm E and can be added to the previous two Poisson's terms. We will call it `Jfix` and each one of these terms is computed and assemble by the function `J_fix_assembly` in Algorithm G.1, note that, apart from the time derivative, the $\frac{1}{2}$ factor from IMR is present in every Jacobian term, hence only after we sum all these terms

```
Jfix = (Kg_phi + Kg_u + Q_phi + Q_u + Q_u_BC)*0.5 + A
```

we multiply by this factor and only then we add `A`. Where we also define

```
KgphiuG = Kg_phi + Kg_u
```

with both `-I` and `G` included, it will be useful to build the residuals next.

5.2.2 Assembling the two Poisson's Residuals

Each one of the two Poisson's residuals has three integrals contributions in (3.36). In Chapter 3 we derived their discretizations, giving us three $5n_q \times 1$ vectors for which we now develop the code for a single residual vector for both potentials.

The first of the three terms is the source term, it requires first to define a new structure for the magnetization. The column vector `x` has each magnetization is at every `x[5*i:5*i+3]` slice, from these we reorganize them into a $3 \times 4n_e$ array `malpha`, where each $3 \times n_e$ block is associated with an α and each column is $\mathbf{m}_\alpha^e$.

$$\texttt{malpha} = \left[\begin{array}{c|c|c|c} \alpha = 0 & \alpha = 1 & \alpha = 2 & \alpha = 3 \end{array} \right]_{3\times 4n_e} \tag{5.22}$$

obtained in A.1. From (3.40) we will require the divergence of the magnetization (3.37) for each element, it is given by the sum of the dot product between the magnetization, $\mathbf{m}_\alpha^e$, with the gradient of the basis function, ∇N_α^e, for the four α, each of these can then be repeated for all e. We can accelerate this procedure by changing what we assume as being fixed. Instead of fixing e and cycling through α we are better fixing α and vectorize the calculations for all e. Therefore we multiply element-wise `malpha[:,a*ne:(a+1)*ne]*Grads[:,a*ne:(a+1)*ne]` for a given $\alpha =$ `a` giving us a $3 \times n_e$ array, if now we summed this array column-wise it would yield $\nabla \cdot (\mathbf{m}_\alpha^e N_\alpha^e)$ for every e, but if we first sum the former for all other α, then we will only need to make a single `sum`

```
divm = sum(malpha0*Grads0+malpha1*Grads1+
           malpha2*Grads2+malpha3*Grads3)
```

giving us a $1 \times n_e$ array with the divergences for every element, (3.37).

To build the first contribution for the residual we will use this `divm` array. Since the divergence only depends on the element e, the calculation in (3.40) is obtained by summing all divergences for all elements that have i as one of its vertices, this is done by `fs_src_phi` function of Algorithm B.2 together with the indices in B.3, giving us an array `s_src_phi_u` with shape $5n_q \times 1$.

The corresponding residual source term for the u potential is exactly the same as the one just computed. But one entry below in the $5n_q \times 1$ vector. Therefore to build a residual vector for both Poisson's problems, we can simply slice each fourth entry at every $5i \times 1$ block and map them into the u entries as `s_src_phi_u[4::5]=s_src_phi_u[3::5]` as we do in `#1` in the following algorithm, B.6

```
#1.The source term.
s_src_phi_u=fs_src_phi(malpha,vol,Grads,nq,barcode)
s_src_phi_u[4::5]=s_src_phi_u[3::5]#copy src term of phi to u entries.
s_src_phi_u[Igs*5+3]=0             #zero the phi boundary entries.
#2.The boundary u term.
s_u_BC=fs_u_BC(malphas,alphas,normal_vecs,areas,nq)
#3.Compute the residual (KgphiuG does not have 0.5, its in avg x)
sphiu=s_src_phi_u+KgphiuG.dot(x)-s_u_BC
```

However the array `s_src_phi_u` still needs further modification, since for every i vertex at the boundary we do not have a Poisson's residual but the residual in (4.60). Hence, for these $5i \times 1$ slices of `s_src_phi_u` we do not have the entries just computed, instead of changing `fs_src_phi` and its indices we simply set them equal to zero as `s_src_phi_u[Igs*5+3]=0`.

The residual at these vertices can be joined with the stiffness residual vector for both potentials and the BEM coupling. This vector term is essentially the `KgphiuG` matrix computed in `#2` times the average of configuration `x=(x+x0)*0.5`, note the 1/2 IMR factor is included on the average configuration and not in the Jacobian itself.

The final integral for the Poisson's residual is only present for the u potential, it is computed by the `f_s_u_BC` function in Algorithm B.5, together with the stiffness terms and residual for the boundary vertices, the residual for the two Poisson's problems the $5n_q \times 1$ column `csr` array `sphiu` that corresponds to `s` from Section 4.6.

5.3 Exchange Matrix Block

We will focus on the calculation of (3.93) and only then substitute IMR. It can be split into two parts. We will develop first

$$\sum_{\beta=0}^{3} \mathbf{m}_{\beta}^{e} \frac{K_{\alpha,\beta}^{e}}{4} \tag{5.23}$$

This is done by the function `Jex` in C.1. We will combine `malpha` structure with the Poisson's entries `Kg` in (5.16). Consider a particular α, then (5.23) is a single sum along the $\beta \in \{0,1,2,3\}$. It can be extended to every e by

$$\frac{1}{4}\sum_{\beta=0}^{3} \begin{bmatrix} m_{x,\beta}^{0} & & m_{x,\beta}^{n_e-1} \\ m_{y,\beta}^{0} & \dots & m_{y,\beta}^{n_e-1} \\ m_{z,\beta}^{0} & & m_{z,\beta}^{n_e-1} \end{bmatrix} * \begin{bmatrix} K_{\alpha,\beta}^{0} & \dots & K_{\alpha,\beta}^{n_e-1} \end{bmatrix} \tag{5.24}$$

with both blocks having n_e columns. The first is a slice of (5.22) given by `malpha[:,b*ne:(b+1)*ne]` and the second, the slice of `Kg` associated with (α,β). We multiply each column of the first by the corresponding entry of the second, giving us a $3 \times n_e$ array which is summed element-wise with the others for $\beta = 1,2,3$. The calculation (5.24) have now to be extended for all α. Starting with a $3 \times 4n_e$ array `B0` with only zero entries

```
B0 = zeros([3,4*ne])
for c in range(16):
    a=ii[c]              #alpha
    b=jj[c]              #beta
    B0[:,a*ne:(a+1)*ne]+=malpha[:,b*ne:(b+1)*ne]*Kg[c*ne:(c+1)*ne]
```

we compute each summand in (5.24) and add it to the right block in `B0` which is guaranteed by the sequences in `ii` and `jj` introduced previously for Poisson blocks. The final result is a `B0` array with four consecutive blocks, each one with shape $3 \times n_e$ and representing the calculation given in (5.24) for a given α.

The second term is given by $\mathbf{m}^e_{tot}\frac{K^e_{\alpha,\gamma}}{4}$. For a given e, each combination `c` in $\{0,1,...,15\}$ determines the $K^e_{\alpha,\gamma}$ while $\mathbf{m}^e_{tot}$ is constant. Defining a `mtot` as $3\times n_e$ array with columns as (3.84) we compute it as Algorithm A.3. Using just the sequence `ii` this single multiplication is extended to all e as `Kg[c*ne:(c+1)*ne]*mtot` which is then added to `B0` sliced at the corresponding α

```
Bvec = zeros([3,16*ne])
for c in range(16):
    a=ii[c]                 #alpha.
    Bvec[:,c*ne:(c+1)*ne]=(B0[:,a*ne:(a+1)*ne]-
                           Kg[c*ne:(c+1)*ne]*mtot)/4
```

the $1/4$ factor in (3.93) is only now introduced since the `B0` term will be used in the exchange residual calculation without it. The final array will be structured as

$$\texttt{Bvec} = \Big[\ (0,0)\ \Big|\ (0,1)\ \Big|\ (0,2)\ \Big|\ \dots\ \Big|\ (\alpha,\beta)\ \Big|\ \dots\ \Big|\ (3,3)\ \Big]_{3\times 16n_e} \tag{5.25}$$

Remember that building the Poisson's matrix required our data aligned in a single row, so that two arrays of indices could map all entries into the final Jacobian matrix. Our array `Bvec` has three rows and also notice we now have to map to a 3×3 matrix block embedded in a 5×5 block with indices (i,j) instead of a single Poisson entry. We will divide the 3×3 block into its lower triangular part and upper triangular, and then build the indices for both, but first `Bvec` has to be transformed into an array with shape $1\times(3*16n_e)$. Notice in (3.75) the entries for the lower triangular part have the y component negative, hence

```
B=zeros([16*ne*3*2,])
#Lower triangular. Flatten and app (-1) to y components
B[:16*ne]           =  Bvec[0,:] #x
B[16*ne:2*16*ne]    = -Bvec[1,:] #y
B[2*16*ne:3*16*ne] =  Bvec[2,:] #z
#Upper triangular has oposite signal.
B[3*16*ne:]         = -B[:3*16*ne]
```

Since its a skew-symmetric matrix block, the upper triangular part is the minus the lower one, we then copy all previous entries, `B[:3*16*ne]` and multiply them by `-1` and add to `B[3*16*ne:]`. The result is a single row array `B` divided into two blocks, and each one has three sub-blocks of length $16n_e$, associated with the x, y and z components and in turn for a given component we have a (α,β) structure with $16n_e$ entries, that is one of the rows of `Bvec`. From another point of view, the idea is to flatten `Bvec` x, y and z rows twice, and "glue" both arrays, the difference between them are the signals, while on the first half the y components have negative sign and x and z have no signs, on the other its the opposite.

What remains to be done from this structure and the structure of matrix $\mathbf{J}$ is to build the corresponding indices for `B` array. We start bottom up and observe that each row of `Bvec` has the same (α,β) structure as `Kg` therefore we build the indices from `alphae` rows similar to what we did for Poisson's term (p. 59).

```
#1.For x row of Bvec build the sequence of
#(alpha (i),gamma (j)) indices
ii=outer([0,1,2,3],[1,1,1,1]).flatten()
jj=outer([1,1,1,1],[0,1,2,3]).flatten()
#2.For a row of Bvec build the (16) block indices
Ig = zeros([16*ne,],dtype='int')
Jg = zeros([16*ne,],dtype='int')
for c in range(16):
    Ig[c*ne:(c+1)*ne]=alphae[ii[c],:]
    Jg[c*ne:(c+1)*ne]=alphae[jj[c],:]
```

Then we copy both `Ig` and `Jg` three times, one for each row of `Bvec`. We multiply it by `5` to yield the locations of the upper left entry of the (i, j) block of 5×5, and then add `+1`, or `+2` or `+3` local fixing depending on whether we are building the indices for the lower triangular or upper triangular. Explicitly we add them as follows

```
IgEx=zeros([3*16*ne*2,],dtype='int')
JgEx=zeros([3*16*ne*2,],dtype='int')
#4.To the 16 block locations in Jacobian,
#add the local x,y,z increments
Igxyzfix=array([2,2,1])
Jgxyzfix=array([1,0,0])
for d in range(3):
    #lower trianglar
    IgEx[d*ne*16:(d+1)*ne*16]=Ig*5+Igxyzfix[d]
    JgEx[d*ne*16:(d+1)*ne*16]=Jg*5+Jgxyzfix[d]
    #upper triangular
    IgEx[(3+d)*ne*16:(3+d+1)*ne*16]=Ig*5+Jgxyzfix[d]
    JgEx[(3+d)*ne*16:(3+d+1)*ne*16]=Jg*5+Igxyzfix[d]
```

Where `Igxyzfix[d]` and `Igxyzfix[d]` entries are with respect to the lower triangular part, for example `d=1` corresponds to a y entry, therefore from (3.75) we need to add `+2` down the `i` direction while `j` direction does not require fixing, hence `+0`. For the upper triangular part we switch them. The final result is three arrays `B`, `IgEx` and `JgEx` which serve as input to `csr_matrix`. The `J` matrix contributions from exchange term is then added to the Poisson's matrix, notice both will be $5n_q \times 5n_q$ matrices and both are pieces of the final Jacobian.

In order to include IMR, this function needs the input variable, `malpha`, to be the average between `malpha0` associated with all $\mathbf{m}^e_{\beta,0}$ and the current guess $\mathbf{m}^e_{\beta,p}$, `malpha`.

5.3.1 Exchange Residual

Observe in (3.85) that the term in parenthesis is the same as the first term in (3.93) which was already computed in `B0`, for a given α and e we compute the cross product of $\mathbf{m}^e_{tot}$ with $\sum_{\beta=0}^{3} \mathbf{m}^e_\beta K^e_{\alpha,\beta}$. We can extend this operation for all elements, using `B0`, since it is already

organized in four blocks of $3 \times n_e$ one for each α. The total magnetization for each element constitutes the $3 \times n_e$ array `mtot`. What we do now is to use these data structures and introduce a vectorized version of the cross product, this is done by noticing that

$$[\mathbf{a} \times \mathbf{b}]_i = \epsilon_{ijk} a_j b_k = a_j b_k - a_k b_j \tag{5.26}$$

implies we can compute the cross product by summing two element-wise multiplications between the vectors $\mathbf{a}$ and $\mathbf{b}$ with the entries permuted as follows

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_0 \end{bmatrix} * \begin{bmatrix} b_2 \\ b_0 \\ b_1 \end{bmatrix} - \begin{bmatrix} a_2 \\ a_0 \\ a_1 \end{bmatrix} * \begin{bmatrix} b_1 \\ b_2 \\ b_0 \end{bmatrix} \tag{5.27}$$

The next lines of code will compute these cross product for $\alpha \in \{0, 1, 2, 3\}$ for all e

```
#1.Cross is computed for each vertex alpha
rvec=zeros([3,4*ne])
for a in range(4):
    x = mtot[[1,2,0],:]*B0[[2,0,1],a*ne:(a+1)*ne] #Permute rows in mtot
    y = mtot[[2,0,1],:]*B0[[1,2,0],a*ne:(a+1)*ne] #and B0 and e-wise mult.
    rvec[:,a*ne:(a+1)*ne] = -(x-y)/4              #-(cross product)/4.
```

Where for example `mtot[[1,2,0],:]` is a $3 \times n_e$ array obtained from `mtot` by flipping the first and second rows and then the new second and third. Similarly `B0[[2,0,1],a*ne:(a+1)*ne]` block has its rows flipped as $[b_2, b_0, b1]^T$ in (5.27). The output is an array with shape $3 \times 4n_e$ where each column is given by (3.86) for a given α.

In order to map these vectors into the final residual vector we have to flatten these three rows in similar manner as we did previously with the exchange Jacobian.

```
#2.Flatning
r_EX=zeros([3*4*ne,])
r_EX[:4*ne]     = rvec[0]
r_EX[4*ne:8*ne] = rvec[1]
r_EX[8*ne:]     = rvec[2]
```

The indices that will map each entry of `rEX` vector into the final $5n_q \times 1$ vector are built by observing the structure of one of the rows of `rvec`, it has the same α block structure as `malpha`. We start by building an array of indices `Ig` for this structure as

```
#1.Define indices for the x component.
Ig=zeros([4*ne,],dtype='int')
for a in range(4):
    Ig[a*ne:(a+1)*ne]=alphae[a]
```

Noticing that `r_Ex` has a structure given by three consecutive flattened `malpha` structures giving us a $1 \times (3 * 4 * n_e)$ array. We modify `Ig` to give us the final indices by multiplying by

5 for the locations of the first entry in each 5×1 block of the final residual, onto which we then add +0 if its an x entry, +1 if its y and +2 if its a z entry depending on which one of the the three blocks of indices we are building.

```
#2.Build indices for each one of the three rows of
#rvec in rEx function.
Igxyzfix=array([0,1,2]) #x, y ,z components
IgEx=zeros([3*4*ne,])
for d in range(3):
    IgEx[d*4*ne:(d+1)*4*ne]=Ig*5+Igxyzfix[d]
```

5.4 Assembling all terms

The final algorithm will assemble all matrix $\mathbf{J}$ and the residual vector $\underline{\mathbf{r}}$ from an initial configuration x0 and a particular one x.

The $\mathbf{J}$ blocks contributions are already available now by the functions previously described and all the other in the Appendix. Hence the remaining task is to simply initiate each of these functions in sequence, but first we have to build their inputs from x and x0, we will require $\mathbf{m}^e_\alpha - \mathbf{m}^e_{\alpha,0}$, and the averages for $\mathbf{m}^e_\alpha$, ϕ^e_α and $\mathbf{m}^e_{tot,\alpha}$, which are computed as follows

```
avg=zeros([3,nq])        #1.Initiate an array with zeros.
dx=x-x0                  #2.Compute the difference between configs.
avg[0]  = dx[0::5]       #3. Put each component of mag at
avg[1]  = dx[1::5]       #one row of avg, the natural
avg[2]  = dx[2::5]       #structure for fmalpha.
dmalpha = fmalpha(avg,alphae)
x=(x+x0)*0.5             #4. Compute the avg (IMR) config and replace x.
avg[0]  = x[0::5]        #5. Build an array where
avg[1]  = x[1::5]        #each column is the mag
avg[2]  = x[2::5]        #at i.
avgmalpha = fmalpha(avg,alphae)   #6. Mag at each (e,alpha).
avgmalphas = fmalphaS(avg,alphas) #7. Mag at each (s,alpha).
avg[0] = happ0[0::3]     #8. Do the same for the cte
avg[1] = happ0[1::3]     #applied field...
avg[2] = happ0[2::3]
avghappalpha = fmalpha(avg,alphae)
avg[0] = x[3::5]         #9...and the potential phi.
avgphialpha = fphialpha(avg[0],alphae)
avgmtot = fmtot(avgmalpha,ne)
avg[0]  = x0[0::5]       #10. Malpha structure for m0.
avg[1]  = x0[1::5]       #No average used.
avg[2]  = x0[2::5]
malpha0 = fmalpha(avg,alphae)
```

where we recycle the array `avg` through the sequence.

Finally on top of `Jfix` that was already loaded from memory we add all other contributions as in Algorithm G.2 yielding the `J` $5n_q \times 5n_q$ matrix and column array `r` that are an essential part for the Newton-Raphson method we revisit on the next and final chapter.

5.5 Newton-Raphson algorithm

From sections 4.6 and 3.2 we will build our code for the algorithm. We will define two simple functions. The `NewtonRaphson` function in Algorithm I.2 starts with the configuration `x0` and the particular configuration `x=x0` and solves a sequence of linear systems to yield the configuration `x` at $t + \Delta t$. And the `Update` function in Algorithm I.1 that simply saves this `x` configuration and restarts the `NewtonRaphson` function for the new `x`. Additionally if the applied magnetic field varies with time that process would also be here introduced, but we will choose to keep it to be constant.

The first iteration of the `while` loop below defines the Jacobian and residual of the linear system of equation (5.5) evaluated at a particular configuration `x` equal to `x0` computed by `Jandresidual`.

```
while 1:
    J,r = Jandresidual(x,x0,hap,hap0,deltat,normal_vecs,
                       alphas,alphae,Igs,alphac,Grads,areas,
                       vol,nq,ne,Jfix,KgphiuG,barcode)
    if  it>it_max or norm(r)<restol:
        return x
    deltax=rILU_AMG_GMRES(J,-array(r).reshape(5*nq,),
                      array(x0),args_phi,args_u,args_gmres)
    x+=deltax
```

The residual norm is then computed and compared against a given tolerance, `restol`. Initially this step is skipped as the residual is higher than the tolerance as we expect from using `x=x0`, meaning that `x0` is not a good approximation for the configuration at $t + \Delta t$. By solving now the $5n_q \times 5n_q$ linear system $\mathbf{J}\Delta\underline{\mathbf{x}} = -\underline{\mathbf{r}}$ using methods we will describe in next section, we correct `x0` towards a better solution. The `while` loop repeats the process, computes `J` and `r` again for the `x0` and `x=x0+deltax` and tests now the norm of the residual. If it passes, we assume `x` as being very close to $\mathbf{m}^*_{\approx}$, $\phi^*_{\approx}$ and $u^*_{\approx}$ that satisfy the residuals being equal to zero. The `while` loop exits and the `NewtonRaphson` function returns the output `x0+deltax` to the `Update` function.

```
def Update(x0,hap0,k_lim,k_save,args):
    #1.For a given time step k.
    for k in range(k_lim):
        #2.Test whether current k is a multiple of k_step to save.
        if k%k_save==0:
            "[Block of code]:save config"
```

```
        #4.Set the first particular configurations as x0.
        x=array(x0)
        x0=Newton_Raphson(x,x0,hap0,hap0,args)
```

which is redefined as the new `x0`. Notice above that all this procedure ocurred for a `k=0` iteration in the `for` loop, while now `k=1` starts with both `x0` and `x` as the previously obtained outputs of `NewtonRaphson`.

The process itself is a sequence of `k_lim` `NewtonRaphson` algorithms each one composed by a sequence of solutions of linear systems of equations that ultimately yield a sequence of increments that correct each `k` iteration's guess, `x=x0`, towards the pair `(x,x0)` that minimizes the residuals.

By saving `x` at each `k_save` iteration, we have the magnetization and potential dynamics.

5.6 A solver of linear systems of equations

Each increment `deltax` is computed by the function `rILU_AMG_GMRES` which consists a Generalized Minimum Residual Method (GMRES) we implemented from [25]. The algorithm is known to suffer from stagnation in the decrease of the residual and a standard solution is to accelarate the convergence by the use of Preconditioners [20]. Since we know the matrix structure we used for the non Poisson blocks the ILU preconditioner that was already available in Python since we observed it was very fast. For the Poisson's blocks of the Jacobian we implemented our version of the Algebraic Multigrid algorithm from [20, 23] and the coarse grid point selection algorithms from [26].

Chapter 6

Time Evolution of a Magnetic System

The algorithms we developed so far for computing and assembling matrices, and our version of Newton-Raphson method that solves the $\mathbf{J}\Delta\underline{\mathbf{x}} = -\underline{\mathbf{r}}$, allow us to compute the time evolution of the system. We will compare our results for magnetization evolution and its Discrete Fourier Transform (DFT) with those for the standard problem given in [27], in order to evaluate the validity of our code and suggest improvements.

6.1 Definition of the standard problem

From [27], the system Ω consists of a flat box with shape $120 \times 120 \times 10\,\mathrm{nm}^3$. This is divided into cubic cells with the desired size, for example $5 \times 5 \times 10\,\mathrm{nm}^3$ for Finite Differences where we associate a magnetization vector to each cell. For the Finite Element Method a $5 \times 5 \times 2.5\,\mathrm{nm}^3$ grainier mesh or a thinner mesh with $2.5 \times 2.5 \times 1\,\mathrm{nm}^3$ is used that is then tessellated into 5 or 6 tetrahedra [28] and to each vertex we associate a magnetization vector.

The problem has now two stages: Relaxation stage and Dynamics stage. The first serves to define the initial configuration for the second. We start with a uniform magnetization $\mathbf{m}_i = (0, 0, 1)^T$ and an external applied magnetic field that is uniform across the system and independent of time given by $\mathbf{h}_{ap} = (65.1, 46.5, 0)^T\,\mathrm{kA/m}$, which normalized by the x component yields $\mathbf{h}_{ap,n} \approx (1, 0.715, 0)^T$. Setting the damping constant to a high value, $\alpha_c = 1$, the evolution of each magnetization to align with the applied field is very fast and precession is supressed, as we want. We stop this stage once the equilibrium configuration is attained and save it for future use. It is suggested in [27] that 5 ns should suffice.

The second stage then takes place, the goal is to record the dynamics of every magnetization vector in Ω starting from the last configuration of the Relaxation stage. We reset the damping constant to the lower value of $\alpha_c = 0.008$ and redirect the applied field to $\mathbf{h}_{ap} = (65.5, 45.9, 0)^T\,\mathrm{kA/m}$, also given by $\mathbf{h}_{ap,n} \approx (1, 0.7, 0)^T$, making a 35° with the x-axis. The update then runs for 20 ns with time step Δt and saving the magnetization configuration at every 5 ps.

In both stages we use parameters for permalloy, the anisotropy of the system is not considered, the saturation magnetization $M_s = 800\,\mathrm{kA/m}$, and exchange constant is $1.3 \times 10^{-11}\,\mathrm{J/m}$.

6.2 Data analysis

The data obtained in the Dynamics stage is analysed using the Discrete Fourier Transform that we briefly summarize in Appendix K.

We start by computing the average magnetization over the entire n_q magnetization vectors in the sample as

$$\langle m_y \rangle = \frac{1}{n_q} \sum_{i=1}^{n_q} m_{y,i} \tag{6.1}$$

and compare with the results from OOMMF micromagnetic simulator whose output is used as standard in [27].

Each magnetization vector in the $z = 0$ plane has its own evolution composed by $N - 1$ steps, hence we have an N entry vector given as $\mathbf{m}_{y,i} = (m_{y,0}, m_{y,1}, \ldots, m_{y,N-1})_i^T$ where the set of all $m_{y,0,i}$ was obtained from the Relaxation stage. It is on each one of these vectors that we apply the DFT to yield a vector of complex coefficients $\mathbf{c}_i$ as in (K.11) where each entry is given by (K.12). There are N coefficients of these that will span frequencies from $f_0 = 0$ up to $f_{N-1} = (N-1)/(N\Delta t)$, where Δt is the time step of recorded data, 5 ns, hence we have $f_k/\text{GHz} \in [0, 200[$. With the complex vectors, $\mathbf{c}_i$, for all $z = 0$ vertices we choose one frequency, f_k and average each amplitude as

$$\langle A_k \rangle = \frac{1}{n_{xy}} \sum_{i=1}^{n_{xy}} A_{k,i} \tag{6.2}$$

where n_{xy} is the number of magnetization vectors in that plane. The set of $\langle A_k \rangle$ is then plotted against f_k. Again the results are compared with the same calculations for the magnetization evolution from OOMMF.

We can go further now and analyse resonance peaks observed in this spectrum. By choosing their particular frequencies we plot the spatial distribution of the amplitudes and phases for each $\mathbf{x}_i$ in $z = 0$ slice of the mesh.

6.3 Numerical Results of Relaxation Stage

To mesh the system Ω we used a BCC lattice. We chose a coarser mesh than suggested in [27] and used $5 \times 5 \times 5\,\text{nm}^3$ cells with a central vertex. Since the geometry is simple we developed our own mesh generator as well as the algorithms for computing the connectivity matrices where all distances were measured in units of l_{EX} (*cf* the fundamental scales in section 2.2).

The initial configuration is given by the magnetization $\mathbf{m}_i = (0, 0, 1)^T$ mentioned previously as well as the the two potentials ϕ_i and u_i at every vertex consistent with this magnetizations field. Hence we have to solve first the coupled Poisson's problem.

After setting up the system for 5000 iterations with time step $\Delta t = 1\,\text{ps}$ measured in units of $(\gamma_e \mu_0 M_s)^{-1}$ we simulated the 5 ns. The following evolution of magnetization y component was obtained

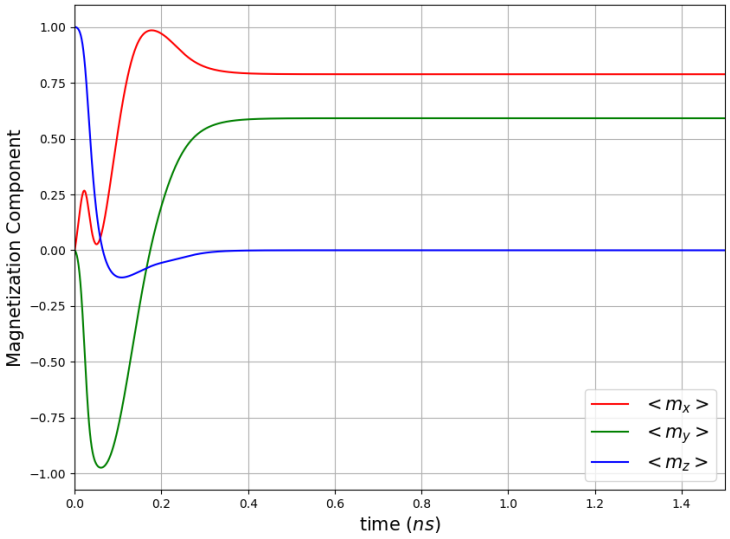

Figure 6.1: Evolution of the average magnetization components of $\langle \mathbf{m} \rangle$ versus time. At 1.5 ns we have approximately $(.7886, .5913, -.0002)$ and at 5 ns we have $(.7884, .5892, -.0008)$

where we only show the first 1.5 ns since the magnetization is approximately constant for the remaining time. We note that the system seems to have reached an equilibrium state after 0.5 ns. Further we note that the norm of magnetization evolves for all 5 ns as

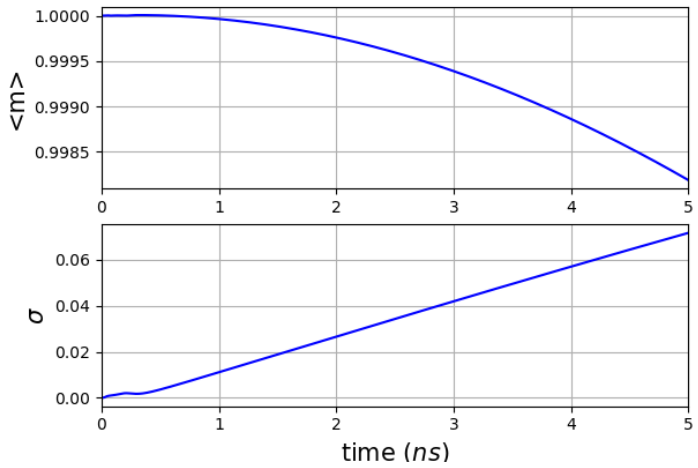

Figure 6.2: Average norm and standard deviation versus time.

meaning that the more iterations, the more we deviate from physically required norm conservation. A spatial plot shows that its the vertices at edges of system that most change their norm. The Dynamics stage will take longer than the first, since this issue will also emerge we decided to start it at a relaxed state but with few norms exceeding 1. Therefore instead of using the configuration at 5 ns, we decided to start the next stage with the configuration at 1.5 ns. The next figure shows a section with $z = 0$ of this data

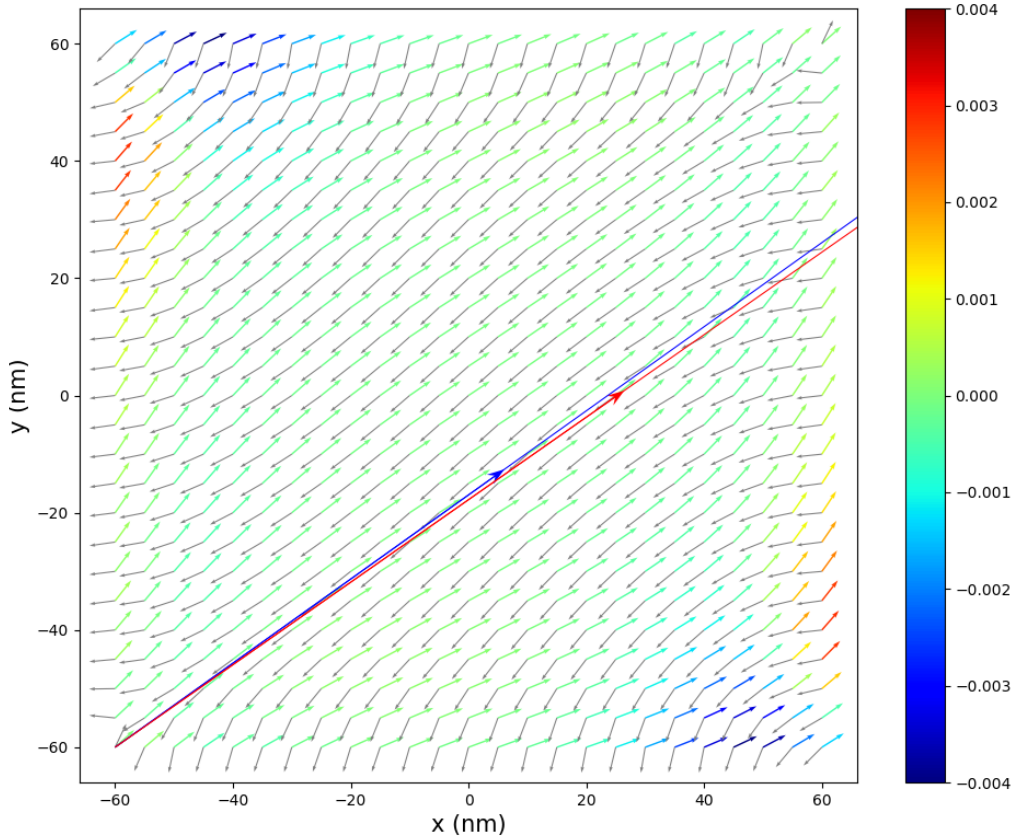

Figure 6.3: Magnetization field for $z = 0$. The colorbar gives the z component. The grey magnetic field belongs to $z = 0$ plane and is the sum of the applied field with the much stronger stray field. The blue and red lines indicates the direction of the applied field for the relaxation and dynamics stages.

In the initial configuration all magnetization vectors point in the z direction and hence we had the four corners had the same local configuration. Upon introducing the applied field, the system evolved into a state where two corners on the left and right of the applied field have magnetization that follows the contour of the boundary and the bottom left and top right almost align with the applied field.

The magnetization are mostly parallel hence the exchange field is small. Plot as a grey vector field is the sum of the applied field with the demagnetizing field, where the latter dominates. Observe the upper and right sides of the system we have $\sigma^* > 0$ while on the lower and left sides the charge density is negative. That is consistent with the direction of the demagnetizing field, it points in the direction opposite of the magnetization. Also notice that on the lower left and upper right sides the direction of $\mathbf{h}_M$ is not what is expected. The demagnetizing field calculation on these two corners might not be accurate enough.

6.4 Numerical Results of the Dynamical Stage

We observed as our simulation proceeded that some of magnetization norms were not conserved. Additionally, since each NR iteration took from 10 s to 1 min, simulating 20 ns at 1 ps steps would require 20000 iterations. Hence we decided to use only data up to 5 ns. The m_y results for the first $2\,ns$ for our code and the OOMMF data from [27] are as follows

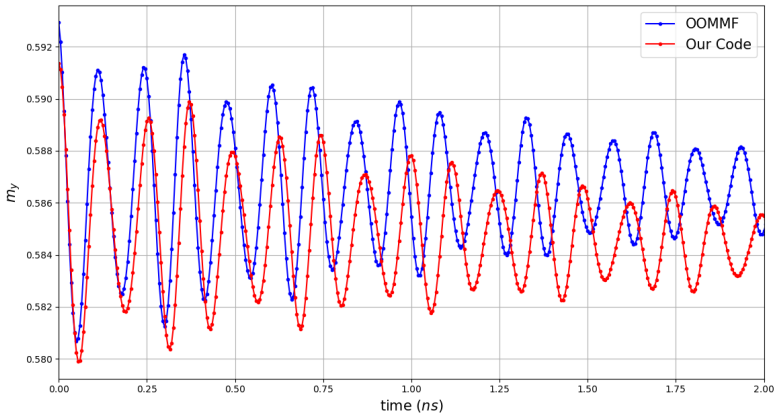

Figure 6.4: Evolution of magnetization y component for $2\,ns$ for our simulation starting at 1.5 ns configuration of the Relaxation stage and OOMMF evolution starting at 5 ns. Initially: $m_y = .591$; $m_y^{\text{OOMMF}} = .593$

The general shape of both waves is very similar. Our results start with magnetization, m_y, 0.02 lower at $t = 0$, which can be caused by the progressive lowering of the magnetization norms as the number of iterations increased as shown in figure 6.2. The peaks are lower in our case and stretched being out of phase after $1.75\,ns$. The evolution of the y component of magnetization from $2\,ns$ onward will keep oscillating with gradually smaller amplitudes but its average also becomes smaller, an aspect not observed in OOMMF whose oscillation is centered on $m_y = 0.586$ for all 20 ns.

The average amplitudes (6.2) of the DFT for each y component of each magnetization in the $z = 0$ plane show the following behaviour in a logarithmic base 10 scale

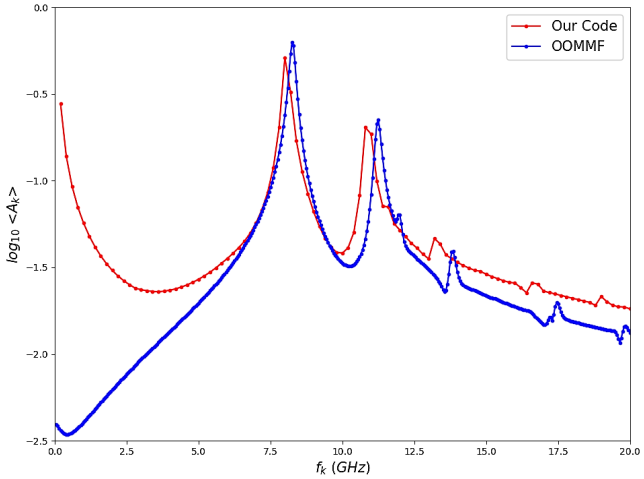

Figure 6.5: Average amplitude for the y components for our simulation and OOMMF in logarithmic scale. The amplitude for $k = 0$ are $10^{-2.7}$ for our results and $10^{-3.3}$ for OOMMF.

The OOMMF results use the full $20\,ns$ giving denser data points than our 5 ns. It was expected from [27] that the bigger is the mesh cell size the higher are the amplitudes for the major peaks, which we do not observe in our results, our two major peaks are slightly lower than OOMMF. The absence of a denser number of data points smoothed out the main

peaks and those smaller ones for frequencies of 12.5 GHz onward. Additionally we observe the presence of very high amplitudes in lower frequencies in figure 6.5, that remains to be explained.

We observe a close agreement for frequencies associated to the two main peaks, one at 8.03 GHz and other at 10.91 GHz, which are shifted from the two peaks for OOMMF given respectively by 8.25 GHz and 11.25 GHz. The cause is associated with the cube dimensions used $5 \times 5 \times 5\,\mathrm{nm}^3$, and the Finite Element Method, as shown in [27] smaller mesh cube would lead to a better approximation to OOMMF results. The shift to the left of our spectrum shows that the major contributors for the evolution of the magnetization y component have smaller frequency, which is consistent with the magnetization evolution being stretched in figure 6.5.

Now, choosing the resonance frequencies detected for the average magnetization we can analyse the spatial distribution of amplitudes and phases for the magnetization evolution in the $z = 0$ plane.

From the frequency of the first resonance 8.03 GHz, we have the following amplitude distribution in logarithmic base 10 scale corrected by the minimum and the corresponding distribution of phases

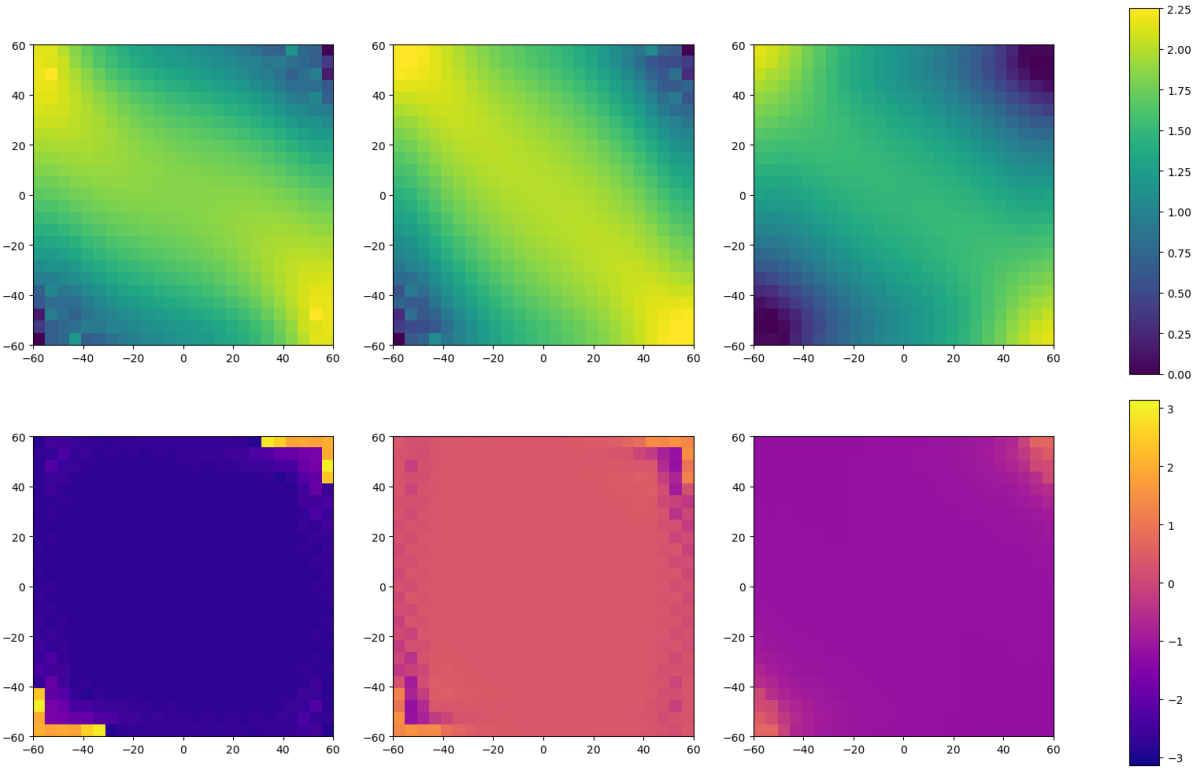

Figure 6.6: Three spatial distribution of amplitudes and phases for x, y and z at 8.03 GHz. Angle between $-\pi$ and π

In the three components we observe for the amplitudes and phases the same symmetry already observed in figure 6.3. Additionally the phases are bigger at two corners coinciding with the smaller amplitudes. The same pattern is observed in the standard results in Appendix J. However figure 6.6 shows for the x and y components that amplitudes and phases at the

lower left and upper right corners are not as smooth as the standard results. This might be caused by the inaccurate stray field calculation as we observed in figure 6.3, according to [27] this field contribution is highly sensitive to the mesh structure. A thinner mesh structure could then improve the results or the code developed in section 4.5 must be improved at these boundary regions.

The second peak at 10.91 GHz exhibits the following distributions of amplitudes and phases

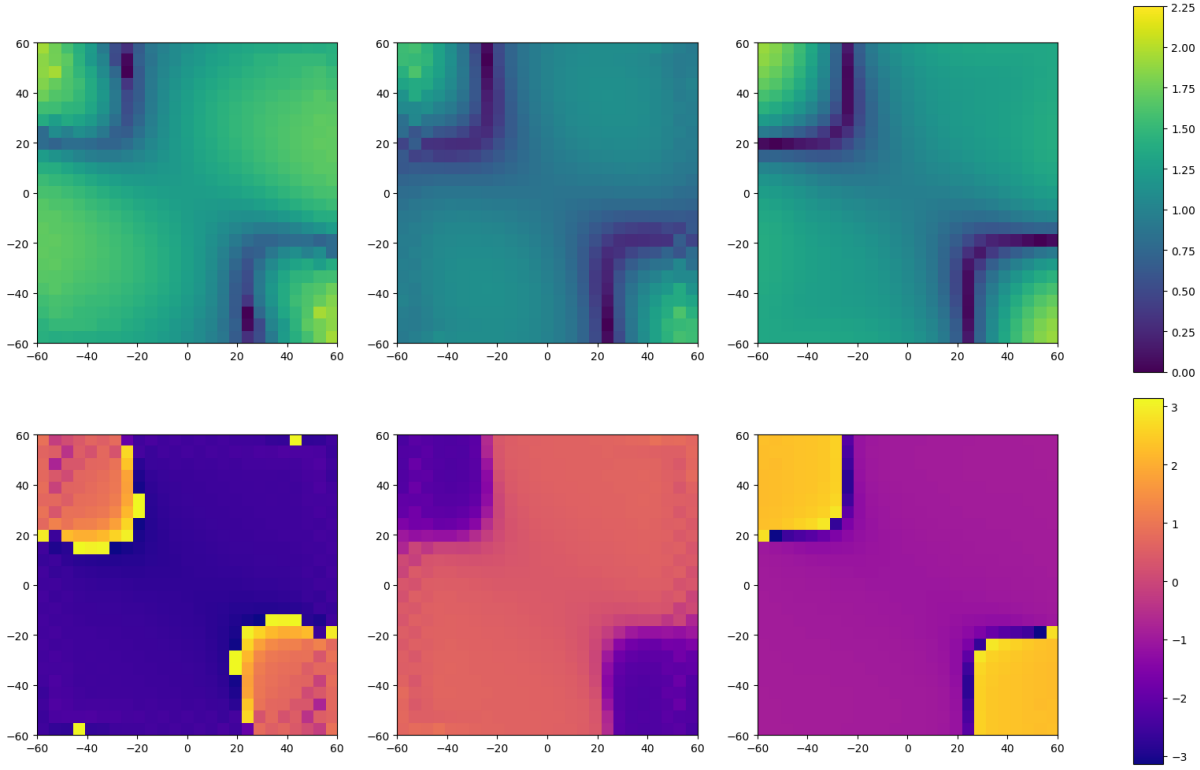

Figure 6.7: Spatial distribution of x, y and z amplitudes and phases for 10.91 GHz.

Like the previous frequency we observe also good agreement between our results and the standard ones in figure J.2, with ours being less smooth. Additionally for the x component phases, we do not observe in figure 6.7 at the center of the system the $\theta \approx \pi$ region we expect from the standard results.

For the x component there is a yellow "shell" enclosing a region with phases near -1 while in z component we have $-\pi$ splitting the regions. This distribution shows domains of phases also observed in figure J.2.

6.5 Conclusion

Our results for the average magnetization evolution and DFT for the y component are in good agreement with the standard problem proposed in [27]. The $\langle m_y \rangle$ wave shows the same oscillations stretched which is consistent with the shift toward lower frequencies in the amplitude spectrum.

The space resolved amplitude and phase are similar to those reported in [27] and shown in Appendix J. The non smooth amplitude and phases near the corners of the system coincide

with magnetization vectors with norm not conserved as the number of iterations increase as well as demagnetizing field having opposite direction to what is expected. Reviewing the code in section 4.5 at the system corners might solve the issue and/or using smaller cubic cells since demagnetizing fields will be more accurately computed.

However the suggestion of reducing the mesh cell structure comes with the price of increasing the number of vertices in the mesh and then the size of the Jacobian matrix and residual vector, meaning larger computational times.

Chapter 7

Conclusion and Outlook

We started with micromagnetic theory that describes the dynamics of magnetization at the microscale. Two equations where introduced: Landau-Lisfshits equation and Landau-Lifshits-Gilbert equation. We rederived the first from the experimental observation that upon application of an external magnetic field magnetization precesses around this field and tends to align with it. From it we develop our intuition for the dynamics implied by each one of the four magnetic field contributions: the applied field; the anisotropy field that benefits certain directions within the system; the stray exchange field which tends to cancel fictitious magnetic charges and the exchange field which smooths out spatial variations in magnetization.

Then we presented the LLG equation derived by Gilbert. Unlike LL, LLG implicitly determines the evolution of magnetization. We then established the goal of developing from scratch a Python code to integrate this equation.

Defining the residuals for LLG and Poisson's equation that determines the stray field, we used Finite Element Method to discretize space and Implicit Midpoint Rule to discretize time. Realizing that the stray field potential requires asymptotic boundary condition and to avoid the computational cost of having to handle the surrounding mesh we opted by a less expensive strategy. We introduced a new potential, solve its Poisson's equation within the system and then use the Boundary Element Method equations to map this potential onto the values of the stray field potential at the boundary. As a consequence the previous asymptotic boundary conditions were replaced by Dirichlet boundary conditions. This rendered integrating LLG equation together with two Poisson's equation using only information about the system, avoiding the need for meshing the surrounding. To solve the equation we make a linear approximation as in Newton Raphson method. The linear system of equations obtained yields as a solution the update for a current configuration of magnetization and potential in the system.

Since this linear system has to be solved at every time step we developed the code to compute the matrix entries using as much as possible vectorized operations giving us fast calculations and assembly.

To validate our code, we implemented the resolution of a standard problem, where we concluded that our results are close to those expected, and can be improved using a denser mesh. During these simulations we observed our solver of linear systems was the slowest step. We implemented our Python version of GMRES with two precondioners, a Python already

available ILU preconditioner for non Poisson's matrix blocks and our version of Algebraic Multigrid for the two Poisson's blocks. Together they revealed to be at least five times faster than the Python sparse matrix solver, being absolute essential in this thesis. However much improvement needs to be done as the successive resolution of linear systems is what makes the code slow, not the actual assembly of matrix and residual vectors. Hence the main focus to improve has to be in using faster methods to solve the linear system of equations.

Despite the high computational time involved, our code has as main feature its simplicity and clear relation with the numerical methods involved in micromagnetism: It serves the purpose of introducing the foundations of numerical micromagnetics, as a source of ideas and as a blueprint for more advanced codes.

Appendices

Appendix A

Data Structures for Magnetization, Potential and Gradients

```
def fmalpha(m,alphae):
    """
    m.shape=(3,nq)
    """
    ne = alphae.shape[1]
    malpha = zeros([3,4*ne])
    malpha[:,:ne]       = m[:, alphae[0,:]]
    malpha[:,ne:2*ne]   = m[:, alphae[1,:]]
    malpha[:,2*ne:3*ne] = m[:, alphae[2,:]]
    malpha[:,3*ne:4*ne] = m[:, alphae[3,:]]
    return malpha
```

Algorithm A.1: The magnetization at each (e, α)

```
def fphialpha(phi,alphae):
    """
    phi.shape=(nq,)
    """
    ne = alphae.shape[1]
    phialpha = zeros([4*ne,])
    phialpha[:ne]       = phi[alphae[0]]
    phialpha[ne:2*ne]   = phi[alphae[1]]
    phialpha[2*ne:3*ne] = phi[alphae[2]]
    phialpha[3*ne:4*ne] = phi[alphae[3]]
    return phialpha
```

Algorithm A.2: The potential at each (e, α)

```
def fmtot(malpha,ne):
```

```
    return (malpha[:,:ne]+malpha[:,ne:2*ne]+
            malpha[:,2*ne:3*ne]+malpha[:,3*ne:4*ne])
```

Algorithm A.3: The total magnetization for each element e.

```
def fgradient(ptot,alphae,signedvol,barcode):
    ne = alphae.shape[1]            #Number of elements=Number of cols.
    C = 1/(6*signedvol)             #1/|X|.

    D_01 = ptot[:,alphae[0]]-ptot[:,alphae[1]]  #Vector differences.
    D_02 = ptot[:,alphae[0]]-ptot[:,alphae[2]]
    D_03 = ptot[:,alphae[0]]-ptot[:,alphae[3]]
    D_12 = ptot[:,alphae[1]]-ptot[:,alphae[2]]
    D_13 = ptot[:,alphae[1]]-ptot[:,alphae[3]]

    G = zeros([3,4*ne]) #Initial G.

    G[:,:ne] = array([(D_12[2,:]*D_13[1,:]-D_12[1,:]*D_13[2,:])*C,
                      (D_12[0,:]*D_13[2,:]-D_12[2,:]*D_13[0,:])*C,
                      (D_12[1,:]*D_13[0,:]-D_12[0,:]*D_13[1,:])*C])

    G[:,ne:2*ne] = array([(D_02[1,:]*D_03[2,:]-D_02[2,:]*D_03[1,:])*C,
                          (D_02[2,:]*D_03[0,:]-D_02[0,:]*D_03[2,:])*C,
                          (D_02[0,:]*D_03[1,:]-D_02[1,:]*D_03[0,:])*C])


    G[:,2*ne:3*ne] = array([(D_01[2,:]*D_03[1,:]-D_01[1,:]*D_03[2,:])*C,
                            (D_01[0,:]*D_03[2,:]-D_01[2,:]*D_03[0,:])*C,
                            (D_01[1,:]*D_03[0,:]-D_01[0,:]*D_03[1,:])*C])

    G[:,3*ne:4*ne] = array([(D_01[1,:]*D_02[2,:]-D_01[2,:]*D_02[1,:])*C,
                            (D_01[2,:]*D_02[0,:]-D_01[0,:]*D_02[2,:])*C,
                            (D_01[0,:]*D_02[1,:]-D_01[1,:]*D_02[0,:])*C])

    with open_file('Gradients%i' %(barcode),'w') as f:
        shape = G.shape
        atom = Float64Atom()
        Grads = f.create_carray(f.root,'Grads',atom,shape)
        Grads[:] = G
        print("File created & saved")
```

Algorithm A.4: The Gradients. Adapted from [24].

Appendix B

Poisson's term

Functions for the source term of Poisson's residual for ϕ and u

```
def fdivm(malpha,Grads,vol):
    """
    IMPUT: To use IMR: malpha -> avgmalpha
    OUTPUT: An array with shape (ne,) with the
    divergences of magnetization at each element e.
    """
    ne = vol.shape[0]
    #1.The gradients for each beta.
    Grads0 = Grads[:,:ne]
    Grads1 = Grads[:,ne:2*ne]
    Grads2 = Grads[:,2*ne:3*ne]
    Grads3 = Grads[:,3*ne:4*ne]
    #2.The magnetizations for each beta.
    malpha0 = malpha[:,:ne]
    malpha1 = malpha[:,ne:2*ne]
    malpha2 = malpha[:,2*ne:3*ne]
    malpha3 = malpha[:,3*ne:4*ne]
    #3.E-wise multiply, e-wise sum, sum column-wise to
    #give a row array with shape=(ne,).
    divm = sum(malpha0*Grads0+malpha1*Grads1+
               malpha2*Grads2+malpha3*Grads3)
    return divm
```

Algorithm B.1: Divergence of magnetization, (3.37).

```
def fs_src_phi(malpha,vol,Grads,nq,barcode):
    """
    The phi Poisson's residual source term contributes with
    this 5nqx1 vector which will be modified in fs_phi_u function
    so that entries in blocks associated with a boundary vertex
```

```
    are zero since we have the s=Gu-phi residual.
    OUTPUT: A csr matrix with shape (5*nq,1).
    """
    ne=vol.shape[0]
    s_src_phi = zeros([4*ne,])
    #1.Divergence of magnetization for every e.
    divm = fdivm(malpha,Grads,vol)*vol/4
    #2.Copy divm for every alpha
    s_src_phi[:ne]       = divm
    s_src_phi[ne:2*ne]   = divm
    s_src_phi[2*ne:3*ne] = divm
    s_src_phi[3*ne:4*ne] = divm
    #3.Load indices and assemble the 5nqx1 column vector.
    with open_file('IgJg_s_src_phi%i' %(barcode),'r') as f:
        Ig = f.root.IgJg_s_src_phi[:,0]
        Jg = f.root.IgJg_s_src_phi[:,1]
    s_src_phi = csr_matrix((s_src_phi,(Ig,Jg)),shape=(5*nq,1))
    return s_src_phi
```

Algorithm B.2: The first term of Poisson's residual, (3.39).

```
def fs_src_phi_indices(alphae,barcode):
    """
    Indices for the first term in Poisson's residual
    for phi. No indices for u are required since we
    will copy s_src_phi in fs_phi_u function for the
    u entries.
    OUTPUT: A file with Ig and Jg as the 1st
    and 2nd cols.
    """
    ne=alphae.shape[1]
    Ig = zeros([4*ne,],dtype='int')
    Jg = zeros([4*ne,],dtype='int')
    Ig[:ne]       = alphae[0]*5+3
    Ig[ne:2*ne]   = alphae[1]*5+3
    Ig[2*ne:3*ne] = alphae[2]*5+3
    Ig[3*ne:4*ne] = alphae[3]*5+3
    with open_file('IgJg_s_src_phi%i' %(barcode), 'w') as f:
        shape=(Ig.shape[0],2)
        atom = Int32Atom()
        IgJg = f.create_carray(f.root,'IgJg_s_src_phi',atom,shape)
        IgJg[:,0] = Ig
        IgJg[:,1] = Jg
```

```
        print("s_src_phi indices saved.")
```

Algorithm B.3: Indices to map each `s_divm` entry into the residual vector.

Functions for the stiffness term of Poisson's residual for ϕ and u

```
def JPoisson(vol,ne,Grads,barcode):
    """
    Stiffness matrix entries for Poisson's residual and Jacobian.
    For the first IMR will not require Kg*0.5 as we multiply
    Kg matrix by the average configuration. If we use Kg
    in a Jacobian we have to multiply Kg*0.5.
    OUTPUT: A file with the array (16*ne,) with all entries.
    """

    G_0 = Grads[:,:ne]
    G_1 = Grads[:,ne:2*ne]
    G_2 = Grads[:,2*ne:3*ne]
    G_3 = Grads[:,3*ne:4*ne]

    Kg = zeros([ne*16,])
                                        #(alpha,beta)
    Kg[0:ne]          = sum(G_0*G_0)*vol   #(0,0)
    Kg[ne:2*ne]       = sum(G_0*G_1)*vol   #(0,1)
    Kg[2*ne:3*ne]     = sum(G_0*G_2)*vol   #(0,2)
    Kg[3*ne:4*ne]     = sum(G_0*G_3)*vol   #(0,3)
    Kg[4*ne:5*ne]     = Kg[ne:2*ne]        #(1,0)
    Kg[5*ne:6*ne]     = sum(G_1*G_1)*vol   #(1,1)
    Kg[6*ne:7*ne]     = sum(G_1*G_2)*vol   #(1,2)
    Kg[7*ne:8*ne]     = sum(G_1*G_3)*vol   #(1,3)
    Kg[8*ne:9*ne]     = Kg[2*ne:3*ne]      #(2,0)
    Kg[9*ne:10*ne]    = Kg[6*ne:7*ne]      #(2,1)
    Kg[10*ne:11*ne]   = sum(G_2*G_2)*vol   #(2,2)
    Kg[11*ne:12*ne]   = sum(G_2*G_3)*vol   #(2,3)
    Kg[12*ne:13*ne]   = Kg[3*ne:4*ne]      #(3,0)
    Kg[13*ne:14*ne]   = Kg[7*ne:8*ne]      #(3,1)
    Kg[14*ne:15*ne]   = Kg[11*ne:12*ne]    #(3,2)
    Kg[15*ne:16*ne]   = sum(G_3*G_3)*vol   #(3,3)

    with open_file('PoissonJacobian%i' %(barcode), 'w') as f: #Save
        shape=Kg.shape
        atom = Float64Atom()
        KgPoisson = f.create_carray(f.root,'Kg',atom,shape)
        KgPoisson[:] = Kg
```

```
        print("File created & saved")

    return Kg
```

Algorithm B.4: Stiffness matrix entries, (3.44). Adapter from [24].

Function for the boundary term of Poisson's residual for the potential u

```
def fs_u_BC(malphas,alphas,normal_vecs,areas,nq):
    """
    Poisson's residual surface integral for the potential u.
    Similar algorithm to BC_Q_u function.
    INPUT:
    malphas is analogous to malpha but comes from using instead alphas.
    malpha.shape=(3,3*ns)
    To use IMR: malphas -> avgmalphas
    OUTPUT:A csr array with shape (5*nq,1). Out of the
    three terms for Poisson's residual, the surface integral
    has a minus sign which is not included in this function.
    Hence we sum the 1st and 2nd terms and subtract this one:
    sphiu=s_divm+Kg.dot(x)-s_u_BC in fs_phi_u function.

    """
    ns=alphas.shape[1]
    BC=zeros([3*3*ns,])
    Ig=zeros([3*3*ns,])
    Jg=zeros([3*3*ns,])
    C=areas/12
    i=0
    kd = lambda a,b: 1 if a==b else 0 #Kronecker Delta fc.
    for b in range(3):
        mbeta = malphas[:,b*ns:(b+1)*ns]
        mbetan = sum(m_b*normal_vecs)*C
        for a in range(3):
            mbetandelta=mbetan*(1+kd(a,b))
            BC[i*ns:(i+1)*ns]=mbetandelta
            Ig[i*ns:(i+1)*ns]=alphas[a]*5+4
            i+=1
    return csr_matrix((BC,(Ig,Jg)),shape=(5*nq,1))
```

Algorithm B.5: Third contribution for Poisson's residual, (3.52).

Residual for Poisson's equation for ϕ and u potential plus the $\mathbf{s}_{\phi}^{\partial} = \mathbf{G}\mathbf{u}_{int} - \boldsymbol{\phi}_{int}$ boundary term

```
def fs_phi_u(KgphiuG,x,malpha,malphas,vol,Grads,Igs,
             nq,alphas,normal_vecs,areas,barcode):
    """
    Poisson's residual for phi and u plus s=Gu-phi.
    INPUT: Kg_phi_[-I] = Stiffness matrix for phi with
    boundary entries replaced by -1.
    KgphiuG -> Kg_phi_[-I] + G + Kg_u.
    x is the current configuration, shape=(5*nq,).
    To use IMR: x, malpha and malphas -> averages.
    malphas is analogous to malpha but uses alphas in its
    construction.
    OUTPUT: A (5*nq,1) csr vector with all Poisson's residuals.
    """
    #1.The source term.
    s_src_phi_u=fs_src_phi(malpha,vol,Grads,nq,barcode)
    s_src_phi_u[4::5]=s_src_phi_u[3::5]#copy src term of phi to u entries.
    s_src_phi_u[Igs*5+3]=0             #zero the phi boundary entries.
    #2.The boundary u term.
    s_u_BC=fs_u_BC(malphas,alphas,normal_vecs,areas,nq)
    #3.Compute the residual (KgphiuG does not have 0.5, its on avg x)
    sphiu=s_src_phi_u+KgphiuG.dot(x)-s_u_BC
    return sphiu
```

Algorithm B.6: Computes the residual for both potentials given by (4.56) for vertices within the system, at boundary by (4.60) and for system plus boundary by (4.64).

Appendix C

Exchange Term

```
def JEx(malpha,mtot,barcode,nq,ne):
    """
    Produces B0 and B flattened. The first
    is used for rexchange, the second in the Jacobian.
    INPUT: malpha -> avgmalpha; mtot -> avgmtot.
    """
    ii=outer([0,1,2,3],[1,1,1,1]).flatten() #Sequence of alphas (ii)
    jj=outer([1,1,1,1],[0,1,2,3]).flatten() #and betas (jj).
    #1.#Load K matrix.
    with open_file('PoissonJacobian%i' %(barcode), 'r') as f:
        Kg=f.root.Kg[:16*ne]
    #2.Compute first term.
    B0 = zeros([3,4*ne])
    for c in range(16):
        a=ii[c]             #alpha.
        b=jj[c]             #beta.
        B0[:,a*ne:(a+1)*ne]+=malpha[:,b*ne:(b+1)*ne]*Kg[c*ne:(c+1)*ne]
    #3.Compute second term and add to the first.
    Bvec = zeros([3,16*ne])
    for c in range(16):
        a=ii[c]               #alpha.
        Bvec[:,c*ne:(c+1)*ne]=(B0[:,a*ne:(a+1)*ne] -
                               Kg[c*ne:(c+1)*ne]*mtot)/4
    #4.Flatten twice.
    B=zeros([16*ne*3*2,])
    #Lower triangular. Flatten and apply (-1) to y components
    B[:16*ne]          =  Bvec[0,:] #x
    B[16*ne:2*16*ne]   = -Bvec[1,:] #y
    B[2*16*ne:3*16*ne] =  Bvec[2,:] #z
    #Upper triangular has oposite signal.
    B[3*16*ne:]        = -B[:3*16*ne]
```

```
    #5.Save.
    with open_file('IgJgJEx%i' %(barcode), 'r') as f:
        Ig=f.root.IgJgJEx[:,0]
        Jg=f.root.IgJgJEx[:,1]
        B=csr_matrix((B,(Ig,Jg)),shape=(5*nq,5*nq))
        return  [B0,B]
```

Algorithm C.1: Exchange contribution for the Jacobian, (3.93).

```
def JEx_indices(alphae,barcode):
    ne=alphae.shape[1]
    #1.For x row of Bvec build the sequence of
    #(alpha (i),gamma (j)) indices
    ii=outer([0,1,2,3],[1,1,1,1]).flatten()
    jj=outer([1,1,1,1],[0,1,2,3]).flatten()
    #2.For a row of Bvec build the (16) block indices
    Ig = zeros([16*ne,],dtype='int')
    Jg = zeros([16*ne,],dtype='int')
    for c in range(16):
        Ig[c*ne:(c+1)*ne]=alphae[ii[c],:]
        Jg[c*ne:(c+1)*ne]=alphae[jj[c],:]
    #3.Build the arrays to receive global indices of every entry in B.
    #The structure of B determines the shape of IgExch and JgExch:
    #3 -> x,y,z, 16*ne -> row lenght of Bvec,
    #2 -> the lower triangular part of 4x4 blocks.
    IgEx=zeros([3*16*ne*2,],dtype='int')
    JgEx=zeros([3*16*ne*2,],dtype='int')
    #4.To the 16 block locations in Jacobian,
    #add the local x,y,z increments.
    Igxyzfix=array([2,2,1])
    Jgxyzfix=array([1,0,0])
    for d in range(3):
        #lower trianglar.
        IgEx[d*ne*16:(d+1)*ne*16]=Ig*5+Igxyzfix[d]
        JgEx[d*ne*16:(d+1)*ne*16]=Jg*5+Jgxyzfix[d]
        #upper triangular.
        IgEx[(3+d)*ne*16:(3+d+1)*ne*16]=Ig*5+Jgxyzfix[d]
        JgEx[(3+d)*ne*16:(3+d+1)*ne*16]=Jg*5+Igxyzfix[d]
    #4.Save.
    with open_file('IgJgJEx%i' %(barcode),'w') as f:
        shape=(IgEx.shape[0],2)
        atom = Int32Atom() #1st col: Ig; 2nd col: Jg
        IgJgJEx = f.create_carray(f.root,'IgJgJEx',atom,shape)
        IgJgJEx[:,0] = IgEx
```

```
        IgJgJEx[:,1] = JgEx
    return IgEx,JgEx
```

Algorithm C.2: Indices for C.1.

```
def rEx(B0,mtot,nq,ne,barcode):
    """
    INPUT: mtot -> avgmtot; B0 is computed by JEx function.
    OUTPUT: a csr matrix with shape=(5*nq,1) of exch residuals.
    """
    #1.Cross is computed for each vertex alpha
    rvec=zeros([3,4*ne])
    for a in range(4):
        x = mtot[[1,2,0],:]*B0[[2,0,1],a*ne:(a+1)*ne] #Permute rows in mtot
        y = mtot[[2,0,1],:]*B0[[1,2,0],a*ne:(a+1)*ne] #and B0 and e-wise mult.
        rvec[:,a*ne:(a+1)*ne] = -(x-y)/4              #-(cross product)/4.
    #2.Flatning
    r_EX=zeros([3*4*ne,])
    r_EX[:4*ne]     = rvec[0]
    r_EX[4*ne:8*ne] = rvec[1]
    r_EX[8*ne:]     = rvec[2]
    #3.Save
    with open_file('IgJgExres%i' %(barcode), 'r') as f:
        Ig_EX = f.root.IgJgExres[:,0]
        Jg_EX = f.root.IgJgExres[:,1]
    return csr_matrix((r_EX,(Ig_EX,Jg_EX)),shape=(5*nq,1))
```

Algorithm C.3: Exchange contribution for the LLG residual.

```
def rEx_indices(alphae,barcode):
    """
    Builds and saves all indices for exchange residual.
    The file is then loaded by rEx function.
    The indices are also the ones required by the rD term.
    """
    #1.Define indices for the x component.
    ne=alphae.shape[1]
    Ig=zeros([4*ne,],dtype='int')
    for a in range(4):
        Ig[a*ne:(a+1)*ne]=alphae[a]
    #2.Build indices for each one of the three rows of
    #rvec in rEx function.
    Igxyzfix=array([0,1,2]) #x, y ,z componets
```

```
    IgEx=zeros([3*4*ne,])
    for d in range(3):
        IgEx[d*4*ne:(d+1)*4*ne]=Ig*5+Igxyzfix[d]
    #3.Save the indices.
    with open_file('IgJgExres%i' %(barcode), 'w') as f:
        shape=(IgEx.shape[0],2)
        atom = Int32Atom() #1st col: IgEx; 2nd col: JgEx.
        IgJgExres = f.create_carray(f.root,'IgJgExres',atom,shape)
        IgJgExres[:,0]=IgEx
        IgJgExres[:,1]=zeros([3*4*ne,],dtype='int')
        print("rEx_indices saved. (also used for rD)")
```

Algorithm C.4: Indices for C.3.

Appendix D

Applied field, Stray Field and Damping

The strategy to compute this Jacobian is to put the three term into arrays with shape $3 \times 4n_e$, four blocks associated with α and n_e vector columns each. We start by element-wise multiply each one of the four blocks of `phialpha` with the corresponding block from `Grads` and add them resulting in a $3 \times n_e$ array, which is then added to each block of `happalpha` and `malpha` yielding a $3 \times 4n_e$ array `Dvec` with four α blocks. It is then flattened twice into a single row with shape $1 \times (2 * 3 * 4 * n_e)$, two halves: one for lower triangular and other with opposite sign for upper triangular. The indices `IgD` and `JgD` are computed starting with the `Ig` indices for a single `Dvec` row. Since the blocks all belong to the main **J** diagonal all we need is `Ig`. We use it repeatedly to build indices for the six blocks of `D` by multiplying all of them with a factor of `5` and then adding the right increments of `+0`, `+1` and `+2` depending on whether its a block associated with x, y or z components on the lower or upper triangular parts.

The residual calculation uses similar strategies as `JD`, the new aspect is the calculation of the cross product already explained in Chapter 5. Observe the indices for `rD` coincide with those for `rEx`.

```
def JD(happalpha,phialpha,deltat,Grads,malpha0,vol,nq,ne,alphac,barcode):
    """
    Builds all entries for the D term.
    INPUT: happalpha and phialpha -> averages
    except malpha0; Grads is the gradient matrix with shape=(3,4*ne);
    vol is the positive volumes of all tetrahedra;
    alphac is the damping constant; barcode identifies the file
    with the indices IgD and JgD to be loaded.
    OUTPUT: A 5*nq by 5*nq csr matrix.
    """
    #1.Define constants.
    vol=vol/4
    cte=alphac/deltat
    #2.Gradient of phi for each element e.
    gradphialpha=zeros([3,ne])
```

```
    for a in range(4):
        gradphialpha+=Grads[:,a*ne:(a+1)*ne]*phialpha[a*ne:(a+1)*ne]
    #3.D vectors to be mapped in skew matrix.
    Dvec=zeros([3,4*ne])
    for a in range(4):
        Dvec[:,a*ne:(a+1)*ne]=(-happalpha[:,a*ne:(a+1)*ne]*0.5
                               +gradphialpha*0.5
                               -cte*malpha0[:,a*ne:(a+1)*ne])*vol
    #4.Flatten Dvec to get the lower triangular part and
    #copy*(-1) for the upper triangular.
    D = zeros([2*3*4*ne,])
    #Lower triangular.
    D[:4*ne]      = Dvec[0] #x
    D[4*ne:8*ne]  =-Dvec[1] #y
    D[8*ne:12*ne] = Dvec[2] #z
    #Upper triangular.
    D[12*ne:]=-D[:12*ne]
    #5.Load indices and build the matrix in csr format.
    with open_file('IgJgJD%i' %(barcode),'r') as f:
        IgD=f.root.IgJgJD[:,0]
        JgD=f.root.IgJgJD[:,1]
    return csr_matrix((D,(IgD,JgD)),shape=(5*nq,5*nq))
```

Algorithm D.1: Applied field, stray field and damping Jacobian.

```
def JD_indices(alphae,ne,barcode):
    """
    All the indices ig,jg for the D term entries
    computed with JD function and saved at 'IgJgJD%i' %(barcode).
    """
    #1.Define indices for the x component.
    Ig=zeros([4*ne,])
    for a in range(4):
        Ig[a*ne:(a+1)*ne]=alphae[a]
    #2.Build indices for other components and final 5*nqx5*nq Jacobian.
    Igxyzfix=array([2,2,1])
    Jgxyzfix=array([1,0,0])
    IgD=zeros([2*3*4*ne,])
    JgD=zeros([2*3*4*ne,])
    for d in range(3):
        #Lower triangular
        IgD[d*4*ne:(d+1)*4*ne]=Ig*5+Igxyzfix[d]
        JgD[d*4*ne:(d+1)*4*ne]=Ig*5+Jgxyzfix[d]
        #Upper triangular
```

```
        IgD[(3+d)*4*ne:(3+d+1)*4*ne]=Ig*5+Jgxyzfix[d]
        JgD[(3+d)*4*ne:(3+d+1)*4*ne]=Ig*5+Igxyzfix[d]
    #3.Save the indices.
    with open_file('IgJgJD%i' %(barcode),'w') as f:
        shape=(IgD.shape[0],2)
        atom=Int32Atom()
        IgJgJD = f.create_carray(f.root,'IgJgJD',atom,shape)
        IgJgJD[:,0] = IgD
        IgJgJD[:,1] = JgD
```

Algorithm D.2: Indices for D.1.

```
def rD(happalpha,Grads,phialpha,malpha,dmalpha,
       alphae,vol,nq,ne,deltat,alphac,barcode):
    """
    Computes the residual vector for the applied field+stray field+
    damping terms of LLG.
    INPUT: happalpha -> avghappalpha; malpha -> avgmalpha;
    phialpha -> avgphialpha; dmalpha = malpha-malpha0 (not avgmalpha)
    OUPUT:A 5nqx1 residual vector is csr format.
    """
    #1.Constants.
    rvec=zeros([3,4*ne])
    vol=vol/4
    cte=alphac/deltat
    #2.Gradient of phi for each element e.
    gradphialpha=zeros([3,ne])
    for a in range(4):
        gradphialpha+=Grads[:,a*ne:(a+1)*ne]*phialpha[a*ne:(a+1)*ne]
    #3.Vectors to be mapped into the residual vector.
    Dvec=zeros([3,4*ne])
    for a in range(4):
        Dvec[:,a*ne:(a+1)*ne]=(+happalpha[:,a*ne:(a+1)*ne]
                               -gradphialpha
                               -cte*dmalpha[:,a*ne:(a+1)*ne])*vol
    #4.Cross product.
    for a in range(4):
        malpha120=malpha[[1,2,0],a*ne:(a+1)*ne]
        malpha201=malpha[[2,0,1],a*ne:(a+1)*ne]
        x=malpha120*Dvec[[2,0,1],a*ne:(a+1)*ne]
        y=malpha201*Dvec[[1,2,0],a*ne:(a+1)*ne]
        rvec[:,a*ne:(a+1)*ne]=x-y
    #5.Flatning.
    r_D=zeros([3*4*ne,])
```

```
    r_D[:4*ne]     = rvec[0]
    r_D[4*ne:8*ne] = rvec[1]
    r_D[8*ne:]     = rvec[2]
    #6.Build the residual vector using IgrD and JgrD,
    #that are the same as IgExchres and JgExchres.
    with open_file('IgJgExres%i' %(barcode),'r') as f:
        IgrD=f.root.IgJgExres[:,0]
        JgrD=f.root.IgJgExres[:,1]
    return csr_matrix((r_D,(IgrD,JgrD)),shape=(5*nq,1))
```

Algorithm D.3: Applied field, stray field and damping contributions for the LLG residual.

Appendix E

P and Q terms

Both terms are computed by cycling through all combinations of (α, β) and evaluating the 3×1 expressions for (3.77) or the first term in (3.56) giving us `Pvec` or `Qvec` arrays with shape $3 \times 16n_e$. These arrays are then flattened giving us `P` and `Q` $1 \times 3 * 16n_e$ arrays. To map these entries into the correct locations within the 5×5 block (5.3) for all blocks in $\mathbf{J}$ we build the indices as follows. Observing `P` and `Q` arrays has a (α, β) 16 entry sequence of indices three consecutive times, each one requiring a local fix corresponding to x, y and z components of the vectors. Since both have the same flattened structure, observing in (5.3) the locations of $\frac{\partial \mathbf{r}_i}{\partial \phi_j}$ and $\frac{\partial s_{\phi,i}}{\partial \mathbf{m}_j}$ are the transpose of one to the other, we conclude that the local transformations given by `IgPxyzfix[d]` and `JgPxyzfix[d]` have to switch roles when obtaining the indices of `Q` from `Ig` and `Jg`.

Since LLG does not depend on the potential u we do not have a P term on the fifth column of a 5×5 block. However, the Poisson's equation for the potential u has a `Q` term that is equal to the one for ϕ but is located one row below, hence all we have to do is use `JgP+1` and `IgP`.

Observe also from the Section 4.6 that at every row block $5i \times 5n_q$ that is associated to a vertex $i \in \partial\Omega$ we do not have a Poisson's residual but (4.60), hence the entries for these rows of `Q` for the potential ϕ have to set equal to zero.

Finally IMR is not implemented in either of `JP_phi` or `JQ_phi`, the `malpha` input has to be equal to `(malpha+malpha0)*0.5`.

For the boundary vertices the entries for $\frac{\partial s_{u,i}}{\partial \mathbf{m}_j}$ have an additional term given by the second term in (3.56) which is computed with `BC_Q_u`.

```
def JP_phi(malpha,Grads,nq,vol,barcode):
    """
    INPUT: malpha -> avgmalpha
    OUTPUT: The P contribution for the Jacobian in csr format.
    To include IMR use avgmalpha and multiply by an extra 1/2
    from the potential.
    The 1/8 factor in vol comes from: 1/2 from avgphi and
    1/4 from basis integral.
    """
    #1.Constants.
```

```
    ii = outer([0,1,2,3],[1,1,1,1]).flatten()
    jj = outer([1,1,1,1],[0,1,2,3]).flatten()
    vol= vol/8
    ne = vol.shape[0]
    #2.Compute the cross procucts between m_a and grad(N_b).
    Pvec=zeros([3,16*ne])
    for c in range(16):
        malpha120 = malpha[[1,2,0],ii[c]*ne:(ii[c]+1)*ne]
        malpha201 = malpha[[2,0,1],ii[c]*ne:(ii[c]+1)*ne]
        a = malpha120*Grads[[2,0,1],jj[c]*ne:(jj[c]+1)*ne]
        b = malpha201*Grads[[1,2,0],jj[c]*ne:(jj[c]+1)*ne]
        Pvec[:,c*ne:(c+1)*ne]=-(a-b)*vol
    #3.Flatten.
    P=zeros([3*16*ne,])
    P[:16*ne]       = Pvec[0] #x
    P[16*ne:32*ne]  = Pvec[1] #y
    P[32*ne:48*ne]  = Pvec[2] #z
    #4.Assemble the 5*nq by 5*nq P matrix.
    with open_file('IgJgJPQindices_phi%i' %(barcode), 'r') as f:
        IgP = f.root.IgJgJPQ_phi[:,0]
        JgP = f.root.IgJgJPQ_phi[:,1]
    return csr_matrix((P,(IgP,JgP)),shape=(5*nq,5*nq))
```

Algorithm E.1: P contribution for the Jacobian, (3.77).

```
def JQ_phi(Grads,vol):
    """
    Computes all entries for the Q term without taking
    into account the surface integral contribuition.
    Notice the output has no minus sign.
    Unlike JP, this term is independent of the configuration.
    To include IMR multiply the output by 1/2.
    OUTPUT: A (3*16*ne,) array with all the entries.
    """
    #1.Constants.
    vol=vol/4
    ne=vol.shape[0]
    jj=outer([1,1,1,1],[0,1,2,3]).flatten()
    #2.For each alpha cycle through all beta's and
    #arrange grad(N_b) in that sequence.
    Qvec=zeros([3,16*ne])
    for c in range(16):
        Qvec[:,c*ne:(c+1)*ne]=Grads[:,jj[c]*ne:(jj[c]+1)*ne]*vol
    #3.Flatten.
```

```
    Q=zeros([3*16*ne,])
    Q[:16*ne]       = Qvec[0] #x
    Q[16*ne:32*ne] = Qvec[1] #y
    Q[32*ne:48*ne] = Qvec[2] #z
    return Q
```

Algorithm E.2: First part of `Q` term, (3.56), for both potential ϕ and u

```
def JPQ_indices_phi(alphae,barcode):
    """
    The indices for P are IgP, JgP. The
    indices for Q_phi are IgQ, JgQ. For first term
    of Q_u use IgQ+1, JgQ. (there no P column for u)
    OUTPUT:A file with an array where the 1st col
    is IgP and the 2nd is JgP, the 3rd is IgQ and
    the 4th is JgQ. All with shape=(3*16*ne,).
    """
    #1.Compute the 16 sequence of alphas and betas
    ne=alphae.shape[1]
    ii = outer([0,1,2,3],[1,1,1,1]).flatten()
    jj = outer([1,1,1,1],[0,1,2,3]).flatten()
    #2.The global vertices indices for one row of Pvec.
    Ig = zeros([16*ne,],dtype='int')
    Jg = zeros([16*ne,],dtype='int')
    for c in range(16):
        Ig[c*ne:(c+1)*ne] = alphae[ii[c], :]
        Jg[c*ne:(c+1)*ne] = alphae[jj[c], :]
    #3.From Ig and Jg fix them to locate the
    #vectorial contribution of P in the 5x5 blocks.
    IgP=zeros([3*16*ne,],dtype='int')
    JgP=zeros([3*16*ne,],dtype='int')
    IgQ=zeros([3*16*ne,],dtype='int')
    JgQ=zeros([3*16*ne,],dtype='int')
    IgPxyzfix=array([0,1,2])    #x, y, z
    JgPxyzfix=array([3,3,3])    #fouth column of 5x5 block.
    for d in range(3):
        IgP[d*16*ne:(d+1)*16*ne]=Ig*5+IgPxyzfix[d] #Local fix
        JgP[d*16*ne:(d+1)*16*ne]=Jg*5+JgPxyzfix[d] #for P indices.
        IgQ[d*16*ne:(d+1)*16*ne]=Ig*5+JgPxyzfix[d] #For Q, switch
        JgQ[d*16*ne:(d+1)*16*ne]=Jg*5+IgPxyzfix[d] #local increments.
    #4.Save indices.
    with open_file('IgJgJPQ_phi%i' %(barcode),'w') as f:
        shape=(IgP.shape[0],4)
        atom = Int32Atom() #1st col: IgP; 2nd col: JgP
```

```
        IgJgJPQ = f.create_carray(f.root,'IgJgJPQ_phi',atom,shape)
        IgJgJPQ[:,0] = IgP
        IgJgJPQ[:,1] = JgP
        IgJgJPQ[:,2] = IgQ
        IgJgJPQ[:,3] = JgQ
        print("JPQ_indices saved.")
```

Algorithm E.3: Indices for both `P` and `Q` terms for ϕ potential, for the u potential there is only the `Q` term.

```
def BC_Q_u(normal_vecs,alphas,areas,nq):
    """
    For every (a,b) comb there is a 3xns array of normal
    vector that have to be mapped. Chosing (a,b) we
    flatten normal_vecs by picking one of its rows
    and build the corresponding indices.
    To include IMR multiply the output by 1/2.
    Similar algorithm to fs_u_BC.
    OUTPUT: A 5*nqx5*nq csr array.
    """
    ns=alphas.shape[1]
    Q_BC=zeros([ns*3*(3*3),])
    Ig=zeros([ns*3*(3*3),])
    Jg=zeros([ns*3*(3*3),])
    C=(areas/12)
    i=0
    kd = lambda a,b: 1 if a==b else 0 #Kronecker Delta fc.
    for a in range(3):
        for b in range(3):
            for x in range(3):
                Q_BC[i*ns:(i+1)*ns]=-normal_vecs[x]*(1+kd(a,b))*C
                Ig[i*ns:(i+1)*ns]=alphas[a]*5+4
                Jg[i*ns:(i+1)*ns]=alphas[b]*5+x
                i+=1
    return csr_matrix((Q_BC,(Ig,Jg)),shape=(5*nq,5*nq))
```

Algorithm E.4: The second term in (3.56) for the u potential.

Appendix F

Time derivative term

The time derivative Jacobian is build starting with a $3 \times 4n_e$ array, `Avec`, where each α block has n_e equal column vectors given by $\frac{\Omega^e}{4\Delta t}(1,1,1)^T$. Then we flatten for each component giving us `A` and build the corresponding indices that map these entries to the main diagonal of the (i,j) block (3.75).

The residual only requires multiplying each α block of `dmalpha=malpha-malpha0` by the n_e volumes. The final structure is the same as the exchange residual contribution.

```
def fA(alphae,vol,deltat):
    vol=vol/(deltat*4)
    Avec=ones([3,4*ne])
    #E-wise multiply each 3xne block of Avec with volumes.
    for a in range(4):
        Avec[:,a*ne:(a+1)*ne]*=vol
    #Flatten.
    A=zeros([3*4*ne,])
    A[:4*ne]      = Avec[0]
    A[4*ne:8*ne]  = Avec[1]
    A[8*ne:12*ne] = Avec[2]
    return A
```

Algorithm F.1: Entries given by (3.64).

```
def JA_indices(alphae,barcode):
    ne=alphae.shape[1]
    IgA=zeros([3*4*ne,],dtype='int')
    for a in range(4):                          #diagonal entry:
        IgA[a*ne:(a+1)*ne]           = alphae[a]*5   #1st
        IgA[4*ne+a*ne:4*ne+(a+1)*ne] = alphae[a]*5+1 #2nd
        IgA[8*ne+a*ne:8*ne+(a+1)*ne] = alphae[a]*5+2 #3rd
    with open_file('IgJgA%i' %(barcode),'w') as f:
        shape=(IgA.shape[0],1)
        atom=Int32Atom()
```

```
        IgJgJA=f.create_carray(f.root,'IgJgJA',atom,shape)
        IgJgJA[:,0]=IgA
        print("JA_indices saved.")
```

Algorithm F.2: Indices for F.1.

```
def rA(dmalpha,deltat,nq,ne,vol,barcode):
    vol = vol/(4*deltat)
    #Multiply each alpha block by the volumes/(4*deltat)
    for a in range(4):
        dmalpha[:,a*ne:(a+1)*ne]*=vol
    #Flatten.
    r_A = zeros([4*ne*3,])
    r_A[:4*ne]     = dmalpha[0,:]
    r_A[4*ne:8*ne] = dmalpha[1,:]
    r_A[8*ne:]     = dmalpha[2,:]
    with open_file('IgJgExres%i' %(barcode),'r') as f:
        Ig_A = f.root.IgJgExres[:,0]
        Jg_A = f.root.IgJgExres[:,1]
    return csr_matrix((r_A,(Ig_A,Jg_A)),shape=(5*nq,1))
```

Algorithm F.3: Residual for the time derivative term, (3.63)

Appendix G

Assembly J and r

```
def J_fix_assembly(barcode):
    #0.Load necessary data.
    #Connectivity matrix.
    with open_file('alphae%i' %(barcode),'r') as f:
        alphae = f.root.alphae[:,:]
    ne = alphae.shape[1]
    #Connectivity array for boundary vertices.
    #Vetices at boundary are Igs.
    with open_file('alphas%i' %(barcode),'r') as f:
        alphas = f.root.alphas[:]
        Igs = f.root.Igs[:]
    ns = alphas.shape[1]
    #Normal vectors.
    with open_file('normalvecs%i' %(barcode),'r') as f:
        normal_vecs = f.root.normalvecs[:]
    #Vertices positions.
    with open_file('allvertexlocations%i' %(barcode),'r') as f:
        ptot = array(f.root.ptot[:])
    ptot = transpose(ptot)
    nq   = ptot.shape[1]
    #Poisson Jacobian for phi. For u potential add +1.
    with open_file('IgJgJPoisson_phi%i' %(barcode),'r') as f:
        Ig_phi = f.root.IgJgJPoi_phi[:,0]
        Jg_phi = f.root.IgJgJPoi_phi[:,1]
    #Q term indices for phi. For u add +1 to IgQ_phi.
    with open_file('IgJgJPQindices_phi%i' %(barcode),'r') as f:
        IgQ_phi = f.root.IgJgJPQ_phi[:,2]
        JgQ_phi = f.root.IgJgJPQ_phi[:,3]
    #Indices for the time derivative Jacobian.
    with open_file('IgJgA%i' %(barcode),'r') as f:
        IgA = f.root.IgJgJA[:,0]
```

```
#Load gradients.
with open_file('Gradients%i' %(barcode),'r') as f:
    Grads = f.root.Grads[:]
#1.Compute the G matrix and save it (Most time consuming step).
boundary_matrix_G(ptot,alphas,Igs,Ngauss,pbox,
                  normal_vecs,k_save,barcode)
#Load G matrix.
with open_file("G_matrix_data%i" %(barcode),'r') as f:
    G = f.root.G[:]
    Ig_G = array(f.root.IgG[:])
    Jg_G = array(f.root.JgG[:])
G=csr_matrix((G,(Ig_G,Jg_G)),shape=(5*nq,5*nq))
#2.Compute Poisson's stiffness matrix (w/o the 1/2 from IMR).
Kg     = JPoisson(vol,ne,Grads,barcode)
Kg_phi = csr_matrix((Kg,(Ig_phi,Jg_phi)),shape=(5*nq,5*nq))
Kg_u   = csr_matrix((Kg,(Ig_phi+1,Jg_phi+1)),shape=(5*nq,5*nq))
#2.Introduce the residual s=Gu-I
Kg_phi[Igs*5+3,Igs*5+3]=-1
Kg_phi+=G
#3.Compute the Q_phi term. (already has the -omega/8 constant)
Q      = JQ_phi(Grads,ne,nq,vol,barcode)
Q_phi  = csr_matrix((Q,(IgQ_phi,JgQ_phi)),shape=(5*nq,5*nq))
Q_u    = csr_matrix((Q,(IgQ_phi+1,JgQ_phi)),shape=(5*nq,5*nq))
Q_BC_u = BC_Q_u(normal_vecs,alphas,areas,nq)
#4.Compute the time derivatice term.
A = fA(alphae,vol,deltat,barcode)
A = csr_matrix((A,(IgA,IgA)),shape=(5*nq,5*nq))
#5.Assemble all constant J terms. Only the Q_u_BC has
#a minus sign, already included in the function.
Jfix = (Kg_phi + Kg_u + Q_phi + Q_u + Q_u_BC)*0.5 + A
#6.Save Jfix.
data    = Jfix.data
indices = Jfix.indices
indptr  = Jfix.indptr
with open_file('Jfix_matrix_data_%i' %(barcode),'w') as f:
    atom = Float64Atom(shape=())
    Jfix_matrix = f.create_vlarray(f.root,'Jfix',atom)
    Jfix_matrix.append(data)
    Jfix_matrix.append(indices)
    Jfix_matrix.append(indptr)
```

Algorithm G.1: This function will build the part of the Jacobian $\mathbf{J}$ given by the time derivative contribution (3.64), the Poisson's stiffness matrix (5.15) for both the ϕ and u and Jacobian with respect to $\mathbf{m}$, (3.56).

```
def Jandresidual(x,x0,happ,happ0,deltat,normal_vecs,alphas,alphae,
                 Igs,alphac,Grads,areas,vol,nq,ne,J,KgphiuG,barcode):
    """
    INPUT: All arguments for all Jacobian and residual functions.
    J is Jfix.
    OUTPUT: A 5nqx5nq csr matrix and a 5nqx1 csr column vector.
    """
    avg=zeros([3,nq])       #1. Initiate an array with zeros.
    dx=x-x0                 #2. Compute the difference between configs.
    avg[0]  = dx[0::5]      #3. Put each component of mag at
    avg[1]  = dx[1::5]      #one row of avg, the natural
    avg[2]  = dx[2::5]      #structure for fmalpha.
    dmalpha = fmalpha(avg,alphae)
    x=(x+x0)*0.5            #4. Compute the avg (IMR) config and replace x.
    avg[0]  = x[0::5]       #5. Build an array where
    avg[1]  = x[1::5]       #each column is the mag
    avg[2]  = x[2::5]       #at i.
    avgmalpha = fmalpha(avg,alphae)   #6. Mag at each (e,alpha).
    avgmalphas = fmalphaS(avg,alphas) #7. Mag at each (s,alpha).
    avg[0] = happ0[0::3]    #8. Do the same for the cte
    avg[1] = happ0[1::3]    #applied field...
    avg[2] = happ0[2::3]
    avghappalpha = fmalpha(avg,alphae)
    avg[0] = x[3::5]        #9...and the potential phi.
    avgphialpha = fphialpha(avg[0],alphae)
    avgmtot = fmtot(avgmalpha,ne)
    avg[0]  = x0[0::5]      #10. Malpha structure for m0.
    avg[1]  = x0[1::5]      #No average used.
    avg[2]  = x0[2::5]
    malpha0 = fmalpha(avg,alphae)

    #Compute all matrix J contributions and residuals.
    J += JD(avghappalpha,avgphialpha,deltat,Grads,
            malpha0,vol,nq,ne,alphac,barcode)
    J += JP_phi(avgmalpha,Grads,nq,vol,barcode)
    r  = rA(dmalpha,deltat,nq,vol,barcode)
    r += rD(avghappalpha,Grads,avgphialpha,avgmalpha,dmalpha,
            alphae,vol,nq,deltat,alphac,barcode)
    r += fs_phi_u(KgphiuG,x,avgmalpha,avgmalphas,vol,Grads,Igs,
                  nq,alphas,normal_vecs,areas,barcode)
    B0,B = JEx(avgmalpha,avgmtot,barcode,nq,ne)
    J += B
```

```
    r += rEx(B0,avgmtot,nq,ne,barcode)
    return J,r
```

Algorithm G.2: Assemble all terms in J and r.

Appendix H

Build G matrix

```
def boundary_matrix_G(ptot,alphas,Igs,Ngauss,pbox,normals,k_save,barcode):
    """
    Compute all integrals and global indices for G matrix.
    INPUT:vertices locations,ptot (shape=(3,nq));
    Boundary connectivity matrix,
    alphas, and the indices of boundary vertices
    Igs (column indices of ptot);
    Ngauss is the number of vertices within a given triangle s used
    in the integration; pbox=[-xbox,+xbox,-ybox,+ybox,-zbox,+zbox].

    OUTPUT:3 arrays saved at "G_matrix_data%i" %(barcode):
    The entries of G (f.root.G) and its 2 corresponding
    global indices for a 5nqx5nq matrix,
    Ig and Jg (f.root.IgG; f.root.JgG).
    """
    "1. Setup up Gaussian locations and weights and the Jacobians"
    epsilon=10**(-15)                  #Tolerance to detect zero integrals
    xi_eta,weights=gauss_triangle(Ngauss)#Ref triangle Gauss locs and weights.
    Jacobians=fJ(alphas,ptot)            #Jacobians from the ref triangles.
    Ig,Jg,G=[],[],[]                     #Where future matrix indices
                                         # and entries will be kept.
    runonce=False #Create file once.
    print("Integrals calculations started.....................")
    "2. Pick a vertex from the boundary and an element
    s and test whether integral is zero"
    for k,i in enumerate(Igs):          #Pick a vertex with
        print("k",k,'out of',len(Igs)-1)#index i from the surface.
        if k%k_save==0:
            if runonce == False:
                with open_file("G_matrix_data%i" %(barcode), 'w') as f:
                    g_matrix = f.create_earray(f.root,
```

```
                'G',Float64Atom(),shape=(0,))
                Ig_g_matrix = f.create_earray(f.root,
                'IgG',Int32Atom(),shape=(0,))
                Jg_g_matrix = f.create_earray(f.root,
                'JgG',Int32Atom(),shape=(0,))
                for g,ig,jg in zip(G,Ig,Jg):
                    g_matrix.append(g)
                    Ig_g_matrix.append(ig)
                    Jg_g_matrix.append(jg)
                #Empty the lists.
                G,Ig,Jg=[],[],[]
                runonce = True
                print("File Created and info saved")
        else:
            with open_file("G_matrix_data%i" %(barcode),'a') as f:
                g_matrix = f.root.G
                Ig_g_matrix = f.root.IgG
                Jg_g_matrix = f.root.JgG
                for g,ig,jg in zip(G,Ig,Jg):
                    g_matrix.append([g])
                    Ig_g_matrix.append([ig])
                    Jg_g_matrix.append([jg])
                #Empty the lists.
                G,Ig,Jg=[],[],[]
                print("saved!")
    x_i = ptot[:,i] #Location of vertex i.
    for s in range(alphas.shape[1]): #Pick a triangular element.
        ig,jg,kg = alphas[:,s] #The 3 vertices locations for s.
        x0s = ptot[:,ig]
        x1s = ptot[:,jg]
        x2s = ptot[:,kg]
        n = normals[:,s]            #The normal to element s.
        d=x0s-x_i                   #The distance from x_i to element s.
        isnormal=dot(n,d)           #If isnormal is zero then skip
        if abs(isnormal) < epsilon:#further calculations.
            continue
        "3. Compute the integrals"
        x0,y0,z0 = x0s      #Build the coordinate transformation
        x1,y1,z1 = x1s      #matrix T from ref triangle
        x2,y2,z2 = x2s      #to triangle element s.
        T = array([[x2,x0-x2,x1-x2],
                   [y2,y0-y2,y1-y2],
                   [z2,z0-z2,z1-z2]])
```

```
            Js = Jacobians[s]    #Pick the corresponding Jacobian.
            for a in range(3):   #For a given Ns_alpha
                I=0              #compute the integral, I.
                for w,(xi,eta) in zip(weights,xi_eta):
                    x = dot(T,array([1,xi,eta]))
                    x = x-x_i
                    x3=4*pi*norm(x)**3
                    f = Ns_a(a,xi,eta)*isnormal*Js/x3
                    I+=w*f

                G.append(-I)     #(-1) from the derivative
                                 #of the Green function.
                j = alphas[a][s] #Append Gij indices fixed by
                Ig.append(i*5+3) #by the Jacobian block structure.
                Jg.append(j*5+4)
    "4. Compute the diagonal terms gamma"
    for i in Igs:
        x_i = ptot[:,i]
        G.append(fgamma(x_i,pbox))
        Ig.append(i*5+3)
        Jg.append(i*5+4)
    with open_file("G_matrix_data%i" %(barcode), 'a') as f:
        g_matrix = f.root.G
        Ig_g_matrix = f.root.IgG
        Jg_g_matrix = f.root.JgG
        for g,ig,jg in zip(G,Ig,Jg):
            g_matrix.append([g])
            Ig_g_matrix.append([ig])
            Jg_g_matrix.append([jg])
        print("Last save!")
```

Algorithm H.1: Algorithm to compute G.

Appendix I

Update Functions

```
def Update(x0,hap0,k_lim,k_save,args,args_phi,
           args_u,args_gmres,barcode,barcode_hist):
    """
    INPUT: x0 is the initial configuration with shape=(5*nq,);
    hap0 is the magnetic field with shape=(3*nq,); k_lim is the number
    of time steps; k_save gives the steps to save x0;
    args is a dictionary with info for Newton_Raphson function;
    args_phi, args_u, args_gmres are the arguments
    for the GMRES algorithm precontioned with AMG for
    phi and u blocks; barcode is the name of the file where we
    saved the mesh info (ex:indices for the residual and J);
    barcode_hist is the name of the file where we save x.
    OUTPUT: a file "xvst%i" %(barcode_hist) with an array
    with shape=(timesteps,5*nq) with timesteps=k_lim%k_save.
    """
    save_j=0      #Current save number=row of "xvst" file.
    runonce=False #To create file once.
    #1.For a given time step k.
    for k in range(k_lim):
        #2.Test whether current k is a multiple of k_step to save.
        if k%k_save==0:
            if runonce==False:
                with open_file("xvst%i" %(barcode_hist),'w') as f:
                    steps=k_lim/k_save
                    shape_x=(steps,5*nq)
                    atom=Float64Atom()
                    history_x=f.create_carray(f.root,'x',atom,shape_x)
                    history_x[save_j]=x0
                    save_j+=1
                    runonce = True
            else:
```

```
                with open_file("xvst%i" %(barcode_hist), 'a') as f:
                    f.root.x[save_j]=x0
                    save_j+=1
        #3.Initiate time counter to predict cumputational time
        #for the k_lim iterations.
        t0=perf_counter()
        #4.Set the first particular configurations as x0.
        x=array(x0)
        x0=Newton_Raphson(x,x0,hap0,hap0,args,args_phi,
                          args_u,args_gmres,barcode)
        t1=perf_counter()
        #5.Print time left until k_lim.
        print("NR #%i started out of %i, took %.4f secs" %(k,k_lim,(t1-t0)))
        print(".............time left: %.4f hrs" %((k_lim-k)*(t1-t0)/3600))
        k += 1
    print("UPDATE FINISHED !!!")
```

Algorithm I.1: The function that computes the evolution of the configuration of the system from initial conditions.

```
def Newton_Raphson(x,x0,hap,hap0,args,args_phi,args_u,args_gmres,barcode):
    """
    INPUT: Arguments decribed in Update function.
    OUTPUT: The congiguration x associated to t+deltat.
    """
    #0.Get variables out of args.
    deltat=args.get('deltat')
    normal_vecs=args.get('normal_vecs')
    alphas=args.get('alphas')
    alphae=args.get('alphae')
    Igs=args.get('Igs')
    alphac=args.get('alphac')
    Grads=args.get('Grads')
    vol=args.get('vol')
    Jfix=args.get('Jfix')
    KgphiuG=args.get('KgphiuG')
    it_max=args.get('it_max')
    restol=args.get('restol')
    #1.Initiate sequence of linear systems.
    it=0
    while 1:
        #2.Build Jacobian matrix and the residual.
        J,r = Jandresidual(x,x0,hap,hap0,deltat,normal_vecs,
                           alphas,alphae,Igs,alphac,Grads,areas,
```

```
                    vol,nq,ne,Jfix,KgphiuG,barcode)
    #3.Compute residual norm.
    normres=norm(r)
    print(".........NR it=",it,"norm res",normres)
    #4.Test whether residual is small enough or no its is exceded.
    if  it>it_max or normres<restol:
        return x
    #5.Solve a linear system of equations.
    x+=rILU_AMG_GMRES(J,-array(r).reshape(5*nq,),
                      array(x0),args_phi,args_u,args_gmres)
    it+=1
```

Algorithm I.2: Our version of the Newton-Raphson algorithm with GMRES with preconditioners ILU and AMG (`rILU_AMG_GMRES` is not described in this thesis).

Appendix J

Amplitude and Phase Distribution for Standard Problem

From the OOMMF magnetization evolution data available from [27] we have the following amplitude and phases distributions for m_x, m_y and m_z for the two resonance peaks: 8.25 GHz and 11.25 GHz.

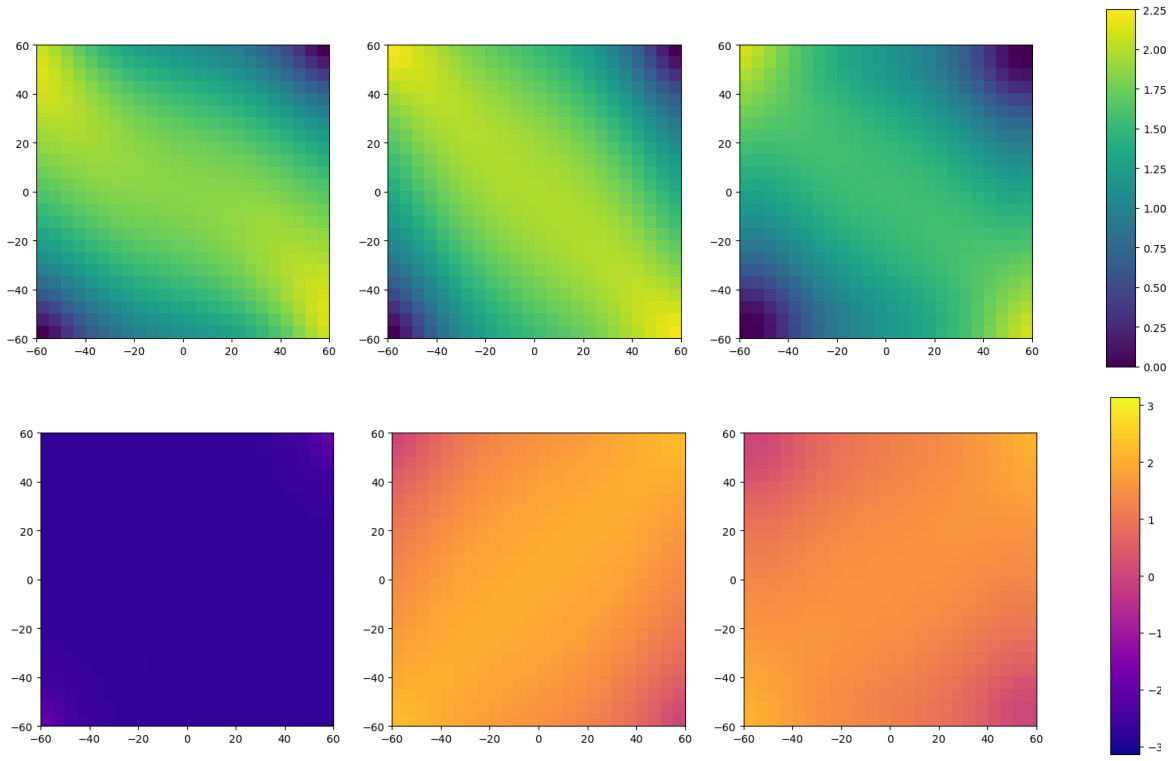

Figure J.1: OOMMF and 8.25 GHz.

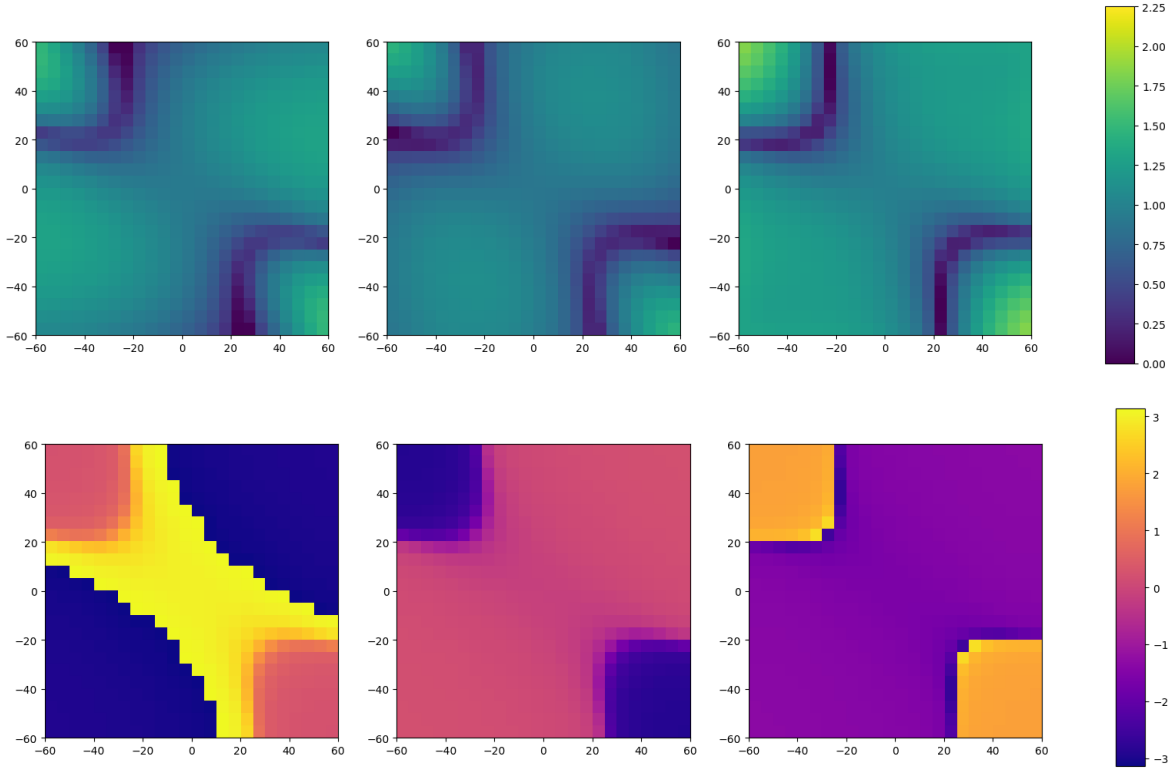

Figure J.2: OOMMF and 11.25 GHz.

Appendix K

Discrete Fourier Transform

We start with the Fourier series and obtain the approximation to the integrals of the coefficients using the trapezoid rule following [12]. Then given them a matrix representation. The goal is to understand what this operation so that the DFT data computed in Chapter 6 is understood.

Consider a periodic function or a slice of a function that is then copied along the x axis. The representation of this function is given by

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{\frac{i2\pi kx}{L}} \tag{K.1}$$

$$c_k = \frac{1}{L}\int_0^L f(x) e^{-\frac{i2\pi kx}{L}} \tag{K.2}$$

In practice the function might be too complicated to integrate or discrete if it results from a laboratory experiment or numerical simulations. This implies that the integrals have to be approximated numerically. For that we can use the Trapezoid Rule [12]. This method has the purpose of approximating the integral as follows. First we divide the a generic interval of integration $[a, b]$ into N segments of index k. The area above each segment A_k is given by the average of the values of $f(x)$ at each point that bounds the segment k times the width of the segment, $h = L/N$. The integrals is then approximately

$$\int_a^b f(x)dx \approx \sum_{e=1}^{N} A_e \tag{K.3}$$

where the higher is the number N the closer is the A_e to the actual area under $f(x)$

$$A_e = h\frac{f(a+(e-1)h) + f(a+eh)}{2} \tag{K.4}$$

The point on the left of segment e is $x_0^e = a+(e-1)h$ while the point on the right is $x_1^e = a+eh$.

Observe that the area above each segment is computed with information associated with each point that determined it. The integral is then given substituting (K.4) into (K.3), isolating the first and last points of $[a, b]$ the integral is

$$\int_a^b f(x)dx \approx h\left[\frac{1}{2}f(a)+\frac{1}{2}f(b)+\sum_{e=1}^{N-1} f(a+eh)\right] \tag{K.5}$$

the sum goes through all points on the right of each segment and that are not on the boundary.

We can use now the result (K.5) to approximate the calculation of the coefficients, we obtain then

$$c_k \approx \frac{h}{L}\left[\frac{f(a)+f(b)}{2}+\sum_{n=1}^{N-1} f(x_n)e^{-\frac{i2\pi k x_n}{L}}\right] \tag{K.6}$$

where n is the index of the point within the $]a,b[$ and $x_n = 0 + nh$. Observe that since $f(x)$ is periodic then $f(0) = f(L)$ hence

$$c_k \approx \frac{1}{N}\sum_{n=0}^{N-1} f(x_n)e^{-\frac{i2\pi k x_n}{L}} = \frac{1}{N}\sum_{n=0}^{N-1} f(x_n)e^{-\frac{i2\pi k n}{N}} \tag{K.7}$$

The first term of the sum $n = 0$ results from the average between the $f(x_n)$ at the two boundary points $x_n = a$ and $x_n = b$.

The calculation is done summing N areas associated with N segments that divide $[0, L]$. Each area depends on two points that define the segment. Since the function is periodic the value of the function at the boundary points is equal hence we lump them leaving $N - 1$ calculation in the sum in (K.5). This average is then introduced in the sum as $n = 0$.

The transformation (K.7) can be used starting with a continuous function which is evaluated at N points within $[x_0, x_{N-1}]$ where $x_N = x_0$ were left or we could start with data already discretized.

With the N coefficients c_k how do we invert the expression and obtain the values of $f(x_k)$ again? Multiplying both sides of (K.7) by $e^{\frac{i2\pi k m}{N}}$ and summing for all k we have

$$\sum_{k=0}^{N-1} c_k e^{\frac{i2\pi k m}{N}} \approx \frac{1}{N}\sum_{n=0}^{N-1} f_n \sum_{k=0}^{N-1} e^{-\frac{i2\pi k(m-n)}{N}} \tag{K.8}$$

where we substituted the definition of h and renamed $f_n = f(x_n)$. On the RHS we observe the second sum is a geometric series that is equal to

$$\sum_{k=0}^{N-1} e^{i2\pi km/N} e^{-i2\pi kn/N} = N\delta_{nm} \tag{K.9}$$

Subtituting in (K.8) we have finally

$$f_m = \sum_{k=0}^{N-1} c_k e^{\frac{i2\pi k m}{N}} \tag{K.10}$$

Observe from the N coefficients c_k we can recover all original discrete $f(x_m)$ values, there is no loss of information.

If we define the fundamental constant $\omega = e^{i2\pi/N}$, we can organize the mapping of coeffi-

cients in (K.10) into all f_m values as the following matrix operation

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \dots & \omega^{(N-1)^2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{pmatrix} \tag{K.11}$$

Observe that (K.10) gives how a particular entry of $\mathbf{f}$ on the LHS of (K.11) is equal to a dot product of a row of the matrix F_N and with the vector $\mathbf{c}$. Notice also the matrix is symmetric. To obtain the vector of coefficients, $\mathbf{c}$, from the values $\mathbf{f}$ we have

$$\mathbf{c} = F_N^{-1}\mathbf{y} \tag{K.12}$$

The key to compute such an inverse is to know the relation among the basis vectors in the columns of F_N. If we choose two columns n and m of F_N and compute the dot product observe what we are actually computing is the expression (K.9). When $n = m$ we get N, the squared norm of the vector, if $n \neq m$ then we have 0, meaning if we normalize all vectors dividing $\frac{1}{N}F_N$ we have a orthonormal basis

$$\frac{1}{\sqrt{N}}\bar{F}_N^T\frac{1}{\sqrt{N}}F_N = I \tag{K.13}$$

where we transpose and take the complex conjugate of the matrix. This matrix relates the basis vectors just as (K.9) and from it we can obtain the inverse yielding

$$F_N^{-1} = \frac{1}{N}\bar{F}_N^T \tag{K.14}$$

Hence expression (K.12) is given by

$$\mathbf{c} = \frac{1}{N}\bar{F}_N\mathbf{f} \tag{K.15}$$

where we used the fact that the matrix is symmetric, $\bar{F}_N^T = \bar{F}_N$. Notice that entry k in $\mathbf{c}$ is given by a dot product of row k with the $\mathbf{f}$ vector, that is the expression (K.7). The meaning of such an operation can better be observed from an example. Suppose $\mathbf{f}$ corresponds to data for N time steps and the variable x is replaced by time, t. Then expression (K.7) can be written as follows

$$c_k \approx \frac{1}{N}\sum_{n=0}^{N-1} f(t_n)e^{-i2\pi f_k t_n} \tag{K.16}$$

where we introduce the following definitions

$$f_k = \frac{k}{N\Delta t} \tag{K.17}$$

$$t_n = n\Delta t \tag{K.18}$$

The sequence of entries in a row corresponds to evaluating $e^{-i2\pi f_k t_n}$ at a sequence of instants of time starting at $t_0 = 0$ up to $t_{N-1} = (N-1)\Delta t$. Its real part or imaginary part represent a

discretized wave with frequency f_k. The bottom rows of $\bar{F}_N$ (or F_N) have higher frequencies that the top ones. Now the vector $\mathbf{f}$ is decomposed in a linear combination of these vectors with coefficients which are complex numbers obtained by computing the dot product, that is by projecting $\mathbf{f}$ into each particular oscillation. The coefficient c_k is characterized by two numbers, the amplitude and phase, qualitatively with both we measure how much $\mathbf{f}$ "looks like" each of these basis vectors.

Appendix L

Software Used

- The code is written using: Anaconda Software Distribution. Computer software. Ver. 4.3.1. Continuum Analytics, Nov. 2017. Web. https://continuum.io.
- The pictures in Chapter 3 and 4 were made with: Harrington, B. *et al* (2004-2005). Inkscape. http://www.inkscape.org/.

Bibliography

[1] Cohen-Tannoudji, Claude, Bernard Diu, and Franck Laloee. Quantum Mechanics. New York: Wiley, 1977.

[2] Blundell, Stephen. Magnetism in Condensed Matter. Oxford: Oxford UP, 2001.

[3] Zangwill, Andrew. Modern Electrodynamics. Cambridge University press, 2012.

[4] Rose-Innes, Alistair Christopher., and E. H. Rhoderick. Introduction to Superconductivity. Oxford: Pergamon, 1969.

[5] Brown, William Fuller. Micromagnetics. New York: Interscience, 1963

[6] Coey, J. M. D. Magnetism and Magnetic Materials. Cambridge: Cambridge UP, 2009.

[7] A.M. Tishin *et all*, A review and new perspectives for the magnetocaloric effect: New materials and local heating and cooling inside the human body, International Journal of Refrigeration, **68**, August 2016, 177–186

[8] L.D. Landau and E.M. Lifshitz. On the theory of the dispersion of magnetic permeability in ferromagnetic bodies, Physik. Zeits. Sowjetunion, **8**, 153–169 (1935).

[9] Gilbert, Thomas L. A Phenomenological Theory of Damping in Ferromagnetic Materials. IEEE Transactions on Magnetics. **40**, 6: 3443–3449

[10] Shepherd, David. Numerical Methods in Micromagnetism, [Thesis]. Manchester, UK: The University of Manchester; 2015.

[11] Bertotti, Giorgio, I. D. Mayergoyz, and Claudio Serpico. Nonlinear Magnetization Dynamics in Nanosystems. Amsterdam: Elsevier, 2009

[12] Newman, M. E. J. Computational Physics, Mark Newman 2013.

[13] Reif, F. Fundamentals of Statistical and Thermal Physics. New York: McGraw-Hill, 1965

[14] Aharoni, Amikam. Introduction to the Theory of Ferromagnetism. Oxford: Clarendon, 1996

[15] Kronmuuller, Helmut, and Manfred Faahnle. Micromagnetism and the Microstructure of Ferromagnetic Solids. New York: Cambridge UP, 2003.

[16] Mallinson, J. "On damped gyromagnetic precession." IEEE Transactions on Magnetics 23.4 (1987): 2003-004.

[17] Strang, G., Computaional Science and Engeneering, Wellesley-Cambridge Press

[18] Goldstein, Herbert, John Safko, and Charles P. Poole. Classical Mechanics. Harlow: Pearson, 2014.

[19] d'Aquino M., Serpico C., Miano, G. Geometrical integration of Landau–Lifshitz–Gilbert equation based on the mid-point rule. Journal of Computational Physics **209** (2005) 730–753

[20] Saad, Y. Iterative Methods for Sparse Linear Systems. Philadelphia: SIAM, 2003.

[21] Jin, Jian-Ming. The Finite Element Method in Electromagnetics. New York: Wiley, 1993.

[22] Elman, Howard C., David J. Silvester, and Andrew J. Wathen. Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics. Oxford, 2005.

[23] Briggs, William L. A Multigrid Tutorial. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1987.

[24] Cuvelier, F., Japhet, C., Scarella, G. (2015). An efficient way to assemble finite element matrices in vector languages. BIT Numerical Mathematics, 56(3), 833-864.

[25] Hanson, J., Kincaid, R. Notes on GMRES Algorithm Organization - UT Computer Science

[26] McCormick, S. F. (1994). Multigrid methods. Philadelphia: Siam.

[27] Baker, A., Beg, M., Ashton, G., Albert, M., Chernyshenko, D., Wang, W., Fangohr, H. (2017). Proposal of a micromagnetic standard problem for ferromagnetic resonance simulations. Journal of Magnetism and Magnetic Materials, 421, 428-439.

[28] Naylor, D. J. (1999). Filling space with tetrahedra. International Journal for Numerical Methods in Engineering, 44(10), 1383-1395

[29] Hunter, P. FEM/BEM NOTES, The University of Auckland

[30] Fredkin, D., Koehler, T. (1990). Hybrid method for computing demagnetizing fields. IEEE Transactions on Magnetics, 26(2), 415-417

[31] Salas. (2006). Calculus: one and several variables. John Wiley

[32] Rathod, H. *et all*, Gauss Legendre quadrature over a triangle, J. Indian Inst. Sci., Sept.–Oct. 2004, 84, 183–188